PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification : Not classified

A2

(11) International Publication Number: WO 98/36630

(43) International Publication Date: 27 August 1998 (27.08.98)

(21) International Application Number: PCT/IL98/00066

(22) International Filing Date: 10 February 1998 (10.02.98)

(30) Priority Data:
60/037,579 10 February 1997 (10.02.97) US

(71) Applicant *(for all designated States except US)*: YISSUM RESEARCH DEVELOPMENT COMPANY OF THE HEBREW UNIVERSITY OF JERUSALEM [IL/IL]; Jabotinsky Street 46, 92182 Jerusalem (IL).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: RAPPOPORT, Ari [IL/IL]; Avraham Granot Street 1, 93706 Jerusalem (IL). SHAMIR, Ariel [IL/IL]; Bar Kochba Street 2, 97875 Jerusalem (IL).

(74) Agent: EITAN, PEARL, LATZER & COHEN-ZEDEK; Lumir House, Maskit Street 22, 46733 Herzelia (IL).

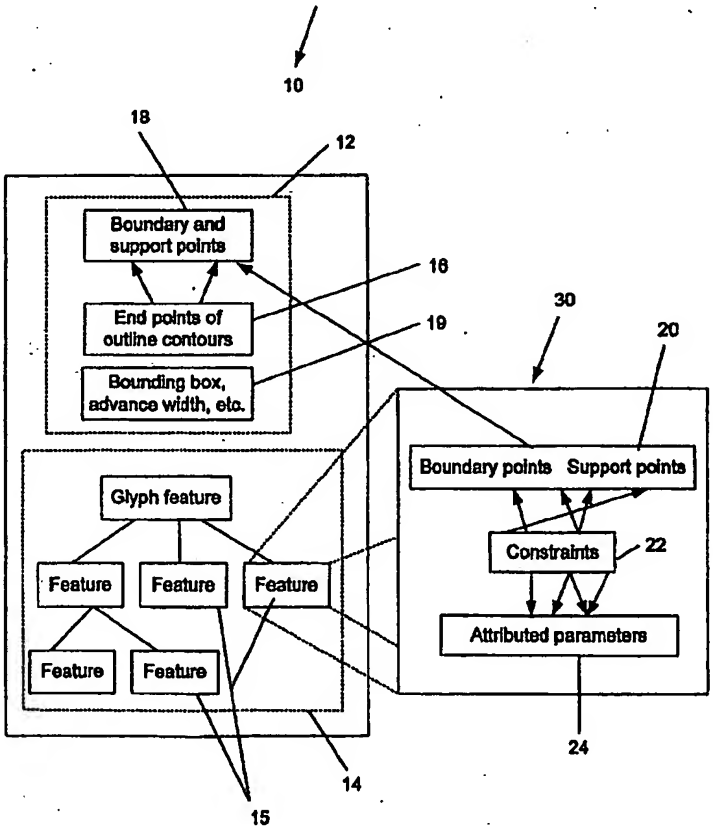
(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, GW, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: PARAMETRIC FONT MODELS BASED ON FEATURES AND CONSTRAINTS

(57) Abstract

A new font model based on parameters, features and constraints is provided. The model supports complex non-linear dynamic behaviors of letter shapes and allows font users to dynamically alter the behavior of letters through the manipulation of external parameters. The letters retain their typographical characteristics during the dynamic behavior. The internal constraints mechanism supports constraint states for continuously handling topological changes. The constraints and features are present in the font model itself. The font model is implemented using parametric features and constraints. The constraints evaluation algorithm operates in linear time in the size of a glyph and guarantees finding a single solution. Constraints are represented using a special novel type of constraints graph, a mixed ratio graph, which can deal with most constraints cycles of the kind arising in the font domain. In addition, a novel constraint state machine enables modification of the shape's underlying topology during dynamic behavior.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

PARAMETRIC FONT MODELS BASED ON FEATURES AND CONSTRAINTS

FIELD OF THE INVENTION

The present invention relates to font models in general and to dynamic font models using geometric constraints in particular.

BACKGROUND OF THE INVENTION

The advent of new digital media such as web pages, CD-ROMs and hypertext files, poses new challenges for computer graphics. The old static page layout is giving way to new and dynamic pages that may change over time or through user interaction. Dynamic HTML, Cascading Style Sheets (CSS), and the Document Object Model (DOM) are but a few of the new standards that create an environment for more dynamic and interactive authoring.

Letters and fonts are extremely important graphics objects in static and even more so in dynamic pages. Unfortunately, current font models do not support dynamic behavior beyond simple affine transformations and interpolations, resulting in lack of visual richness and possible damage to the typographic nature of the design.

The main widespread font models, Postscript Type 1 by Adobe Systems Inc of California, USA and TrueType by Microsoft Corporation Inc. of Seattle, USA, are outline-based and directly support only scaling and rasterization. Dynamic behaviours for fonts are mostly created by recording pictures of text and using them in a movie-style playback, or by simple scripted animations (implemented, e.g., as Java applets). These techniques support dynamic changes of environment attributes such as layer, movement, color, spacing and kerning. The only font attribute that can be modified is size. This reflects the underlying font model, which is a scaleable outline font.

Internal shape deformations of letters are enabled by converting them to general graphical entities and applying general shape deformations. Such entities might be 3-D objects, such as available in "Inventor" by Silicon Graphics of California USA and "Typestry" by Pixar Animations of California, USA or 2-D

objects available in the "Illustrator" and "Freehand" programs of the Adobe Systems Inc. of California, USA, or bitmaps (in paint systems). The changes applied can be 3-D or 2-D transformations, coloring, free-form deformations etc. The major drawback in such techniques is that the letters lose their essence; letters possess a well-defined structure, and we expect it to be preserved during dynamic behavior.

More advanced font models, such as Apple's QuickDraw GX (Apple Computer Inc. of California, USA) and Adobe's Multiple Master fonts (Adobe Systems Inc. of California, USA), directly support linear interpolations of the points defining glyph outlines. The design of such multiple outline fonts is extremely difficult. In addition, the range of shape deformations that can be achieved is limited.

Parametric font models can in principle produce several variations of the same glyph. Most of these were created with selection and compression of static font in mind and do not tackle dynamic issues. The most important model in this category is metafont, which allows attaching parameters to programmable glyph descriptions and also includes an evaluator for algebraic equations. The use of metafont requires programming skills that are not common with graphics designers. In a sense, metafont is more a general purpose graphical programming language than a high-level font model. US Patent No. 5,586,241 to Bauermeister et al. describes a system for parametrically generating characters and fonts.

Many graphics design systems use geometric constraints, such as the use of letter design by Heydon. A, and Nelson, G., in The Juno-2 Constraint Based Drawing Editor, SRC Research Report 131, Digital Systems Research Center, December 1994. Other systems were targeted specifically for font design such as the oriental character design described by Lim, S.B., Kim M.S., in Oriental Character Font Design by a Structured Composition of Stroke Elements, Computer-Aided Design, Vol. 27 No. 3, 1995, 193-207. Such systems do not allow the designer to export the designed font in a way that would let others create dynamic behavior. The exported fonts are stored using the familiar outline

representations, and the constraints are not part of the font but exist only as a tool in the design environment.

High quality rasterization of outline fonts involves grid fitting constraints called 'hints'. Examples of systems which manipulate control points are described in US Patents Nos. 5,155,805 and 5,325,805, both to Kaasila. The two common font standards Postscript Type 1 and TrueType support incorporation of these hints into the font. There are systems that allow adding hints to fonts interactively, such as "Visual TrueType" by Microsoft Corporation Inc. The constraints here are not general geometric constraints and are only used in the context of grid fitting outline glyphs.

SUMMARY OF THE PRESENT INVENTION

It is an object of the present invention to provide a novel dynamic font model which overcomes the disadvantages of existing font models.

The dynamic font model known as "LiveType", supports complex non-linear dynamic behaviours of letter shapes. Inherent typographic and stylistic constraints are preserved during dynamic behavior. The capability of modifying letter shapes through parameter specification is present in the font itself and is provided to any user of the font.

LiveType is implemented using parametric features and constraints. The constraints evaluation algorithm operates in linear time in the size of a glyph and guarantees finding a single solution. Constraints are represented using a special novel type of constraints graph, a mixed ratio graph, which can deal with most constraints cycles of the kind arising in the font domain. In addition, a novel constraint state machine enables modification of the shape's underlying topology during dynamic behavior.

LiveType features combine geometric elements and are usually used to represent typographically significant entities such as bars and serifs. Feature characteristics are preserved using geometric constraints defined between their elements. Features and constraints possess parameters, which can be combined hierarchically in order to encapsulate a variety of complex glyph behaviours. Specification of new parameter values triggers a constraint re-evaluation, which in turn yields deformations of letter shapes. To facilitate interfacing with existing font rasterizers, the LiveType model is capable of rapidly producing glyph outlines.

The constraints evaluation algorithm operates in linear time with respect to the number of points in the glyph and guarantees finding a single solution. The present invention uses the mixed ratio graph, that is a directed graph that can be separated into two directed acyclic graphs according to constraint types that are common in the typographic domain. Secondly, we present the notion of a constraint state machine, which enables modification of the shape's underlying topology during dynamic behavior, in addition to the non-linear geometric

modifications. The state machine allows a geometric entity to change the constraints governing it under certain conditions.

In accordance with an embodiment of the present invention, there is thus provided a glyph which is defined by its geometry and includes a plurality of features each of the features being defined by at least one constraint and at least one parameter attached to at least one of the plurality of features.

Additionally, in accordance with an embodiment of the present invention, there is thus also provided a font model which includes a plurality of glyphs. Each of the plurality of glyphs are defined by its geometry and includes a plurality of features wherein each of the features is defined by at least one constraint.

Furthermore, in accordance with an embodiment of the present invention, the font model further includes a feature store for storing a plurality of glyph features wherein each glyph points to at least one of the stored glyph features.

Additionally, in accordance with an embodiment of the present invention, at least one of the plurality of features within each of at least two different glyphs is connected to a single parameter, whereby by changing the single parameter at least one of the plurality of features within each of at least two different glyphs is correspondingly changed.

Furthermore, in accordance with an embodiment of the present invention, each of the plurality of features are further defined by its boundary and support points.

Additionally, in accordance with an embodiment of the present invention, the plurality of features includes typographic features and supporting features. The typographic features include at least one of a group including bars, stems, bows, arc, curve stems, curve bars and serifs.

Furthermore, in accordance with an embodiment of the present invention, at least two of the plurality of features are treated as a single stroke unit.

Furthermore, in accordance with an embodiment of the present invention, the constraint includes at least one of a group including distances between points and lines, ratios of the distances and angles between lines.

In addition, in accordance with an embodiment of the present invention, each of the plurality of features includes a plurality of parameters combined together to form a typographical axis parameter whereby by changing a single axis value, at least two of the plurality of parameters are correspondingly changed. The rate of change of one of the two parameters is independent of the rate of change of the second parameter.

Furthermore, in accordance with an embodiment of the present invention, a change to each of the parameters is connected to at least one of the constraints. The constraint has a value which is a multiplication of a scalar and the parameter.

Furthermore, in accordance with an embodiment of the present invention, at least two of the plurality of features are connected to a single parameter, whereby by changing the single parameter the two features are correspondingly changed.

Additionally, in accordance with an embodiment of the present invention, there is thus also provided a method for creating a variation of a glyph having at least one constraint. The method includes the step of altering at least one parameter attached to the at least one constraint.

Furthermore, in accordance with an embodiment of the present invention, the method further includes the step of reevaluating the constraints graph of the glyph to modify the geometry the glyph.

Additionally, in accordance with an embodiment of the present invention, there is thus also provided a method for reevaluating a constraint graph for a glyph. The glyph has a plurality of points, each of which has first and second constraints, the graph having outline YRatio and XRatio constraints and a plurality of geometric constraints. The method includes the steps of:

marking all of the plurality of glyph points as unevaluated;

removing the YRatio constraints connected to outline points;

determining the position of each of the plurality of glyph points by evaluating each of its geometric constraints;

marking all of the plurality of glyph points as unevaluated;

removing the XRatio constraints connected to outline points; and

determining the position of each of the plurality of glyph points by evaluating each of its geometric constraints;

Furthermore, in accordance with an embodiment of the present invention, the step of evaluating includes the steps of:

If a point is marked as evaluated, returning its current position;

If a point is marked as un-evaluated, obtaining the first and second constraints;

calculating the first (c1) and second (c2) constraints;

computing the lines of freedom (LOF) for each of the first and second constraints;

If both the first and second constraints are empty, returning the current position;

If the first constraint is empty and if the second constraint is an X constraint, then c1 = Yabs; otherwise c1 = Xabs;

If the second constraint is empty and if the first constraint is an X constraint, then c2 = Yabs; otherwise c2 = Xabs; and

setting the new position of the point to the intersection of the LOF of the first and second constraint.

Finally, in accordance with an embodiment of the present invention, there is also provided a system of constraint states for a geometric object includes a plurality of states wherein each of the states has at least one constraint attached thereto and wherein a change from one of the plurality of states to another is determined by transition rules. The geometric object may be a glyph. The transition rules include any of a group of rules including geometric conditions or time.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be understood and appreciated more fully from the following detailed description taken in conjunction with the drawings in which:

Fig. 1 is a high level block diagram illustration of the dynamic font model (glyph), constructed in accordance with a preferred embodiment of the present invention;

Figs. 2A, 2B and 2C illustrate the outline, features and underlying geometry, respectively, of a Kanji symbol;

Fig. 3A illustrates the outline and supporting constructs of a Courier `B' glyph;

Fig. 3B illustrates the points defining some of the features of the glyph of Fig. 3A;

Figs. 4A and 4B illustrate the effect of changing a parameter on the shape of a glyph;

Fig. 5 illustrates a parametric space of 'A' shapes formed by two parameters and a dynamic path in it;

Figs. 6A-6D illustrate the connection of one parameter to several constraints;

Figs. 7A and 7B illustrate the carrying over a parametric change from one feature to another;

Figs. 8A-8D illustrate the generation of a bolding axis for the `B' glyph;

Figs. 9A-9C illustrate degrees of freedom and line of freedom of a glyph point;

Figs. 10A-10C illustrate constraint types associated with the glyph of Fig. 1;

Figs. 11A -11B illustrates a bar feature from the letter 'A' and the sub-graph of constraints imposed therein;

Figs 12A-12C illustrates the effect of ratio constraints;

Fig. 13A illustrates a simple circle shape;

Fig. 13B illustrates the mixed ratio constraint graph imposed by the shape of Fig. 13A;

Fig. 14 illustrates the underlying geometry of the letter 'A';

Fig. 15A illustrates the pseudo source code of the constraint evaluation algorithm;

Figs 15B is a high level flow chart illustration of the method for re-evaluating a mixed ratio constraints graph whenever a parameter is altered;

Fig. 15C is a high level flow chart illustration of the algorithm of Fig. 15A;

Figs. 16A-E illustrates the bolding of the Courier Font letter 'M' without the use of constraint states;

Figs. 17A-F illustrate the use of constraint states to resolve the topological problems of the letter 'M' of Fig. 16;

Figs. 18, 19, 20 and 21 illustrate alternative applications utilizing the glyph of Fig.1;

Figs. 22, 23 and 24 illustrate possible variations which may be made to existing font models using the glyph of Fig. 1;

Fig. 25 is a high level block diagram illustration of a dynamic font model (glyph), constructed in accordance with a further preferred embodiment of the present invention; and

Figs. 26 and 27 are examples of possible variations illustrating the use of the glyph of Fig. 25.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

Reference is now made to Fig. 1 which is a high level block diagram illustration of a glyph model, generally designated 10, hereinafter also referred to as "LiveType", in accordance with a preferred embodiment of the present invention.

The components of the basic glyph model 10 are its underlying geometry, generally designated 12 and its feature hierarchy, generally designated 14. The geometry 12 includes both the outline 16 and support geometries 18 and 19, which are defined from line segments and curve spleens.

The feature hierarchy 14 comprises a plurality of features 15. Each of the features 15 comprises components (illustrated by the callout box, referenced 30) consisting of its geometry (boundary and support points) 20, constraints 22 and parameters 24. Any subset of the glyph geometry can be designated as a feature 15. Generally, there are typographic features such as bars, stems, bows, and serifs, and supporting features such as the baseline or centerline. Features are organized in a hierarchy, for example, treating a bar and an attached serif as a single stroke unit. By way of example, Figs. 2A -2C illustrate the outline (Fig. 2A), features (Fig. 2B) and complete underlying geometry (Fig. 2C) of a Japanese Kanji symbol. The geometrical features of Fig. 2B include the stems 26 (dark), bars 28 and serifs 30.

Constraints 22 are defined using points belonging to the feature's geometry. There are several constraint types, including distances between points and lines, ratios of such distances, and angles between lines as will be described hereinbelow. The constraints 22 are represented in a constraint graph, which is solved using a constructive evaluation algorithm, to be described hereinbelow. Almost every constraint can be linked to a parameter 24; for example, in a constraint specifying that the distance between two points should be "d", "d' can be used as a parameter. Several parameters can be combined together to form an axis, providing an efficient mechanism for mapping high level font or glyph

parameters and behaviours to low level feature parameters as will be described hereinbelow.

The dynamic functionality of the model is provided through modifications of axes and parametric values. When these are modified, the constraint solver is invoked to compute the new coordinates of the supporting and outline points, and new glyph shapes are produced. Therefore, in order to create continuous change in shape, the user need only apply a continuous change to parametric values.

Similarly to existing outline representations, "LiveType" support points, lines, circular arcs and spleen curves. All these geometric types are represented using 2-D points. Points are classified as being either outline points or support points (e.g., for modeling guiding lines and bounding boxes). Outline points can be further classified as being on-points or off-points, depending on whether they lie on the glyph boundary or not. For example, some spleen control points are off points.

Outline and support points 20 are used in the definition of features inside a glyph. A feature is designated as any subset of the glyph's points. The subset is stored in the feature as an array of point indices. Consequently, a point can be shared by several features, and the features themselves form a cover to the glyph geometry and not a disjoint decomposition. Some examples of features include bars, stems, arcs, bows and serifs. However, features do not have to directly correspond to typographic elements. Any part considered as significant can be designated as a feature.

The dynamic font model 10 of the present application utilizes features 15 to encapsulate functionality of glyph sub-parts and connect them at the font level. For instance, a set of points is called a stem when they are ordered as two facing sets of points having the same distance. This information is important and meaningful especially when it is desired to change the positions of points and change the shape of the design. The fact that these points form a stem will mean that they must remain parallel and in a predefined distance from a centerline during all changes. This distance can be parameterized and, for example, named 'stem width'. The set of constraints needed to preserve these relations can be

defined on the points without defining a feature. However, generally, most features constantly reappear in many glyphs. Thus, it is an advantage to be able to refer to them together and control them as a single logical entity; for example, by modifying all of their stem width parameters described hereinbelow with respect to Figs. 4 and 5.

There are two general methods to define features: design by features and feature recognition. The latter is essential in order to deal with existing fonts. Some key typographic features can be automatically recognized. Other features can be defined interactively by using visual design systems. Features are organized in a hierarchy in order to allow designers to manipulate several elements at once. In practice, the hierarchy is mostly used for grouping serifs to the elements they are attached to, and the feature hierarchies are at most three levels deep: the glyph itself, an element, and its serif(s).

Fig. 3A shows the outline and supporting constructs of a Courier `B' glyph. Each feature includes both a partial outline and supporting constructs, visualized as a line. Lines 1-3 represent bars, 4 the glyph's baseline, 5-6 serifs, 7 a stem, 8 a joint, and 9-10 bows. Line 11 is used as a supporting construct in the definition of the Italic version of the glyph. Fig. 3B shows the points defining some of the features. Squares (for example, 5a, 5b and 9b) denote on-points while circles (9c and 9d) denote off-points.

Any numerical value used in the definition of a constraint such as distance, ratio and angle, can be connected to a parameter. This means that the fixed numerical value is turned into a variable. Parameters are associated with features, as features encapsulate the basic functionality parts of glyphs.

Parameters act as the external interface to dynamic behaviours of the features and glyphs. Thus, modifying a parametric value in a certain feature will cause the constraints it is connected to, to become invalid, and trigger the constraint evaluation process. This will create a new glyph variation.

Reference is now made to Figs. 4A and 4B which illustrate the effect of changing a parameter on the shape. The line distance width is altered from 44 (Fig 4A) to 62 (Fig. 4B).

The dynamic parametric change can described as a navigation in some space of feasible objects. Each parameter defined in a glyph adds a dimension to this space. For example, as shown in Fig. 5 which illustrates a parametric space of 'A' shapes formed by two parameters and a dynamic path in it. Each value pair (horizontal bar-width, top-angle) is mapped to a point in a two dimensional space of possible shapes for the letter `A'. In this context, a dynamic behavior of a letter is in fact a path in the parametric space, and is achieved by changing the vector of parametric values.

In a simple scenario, each parameter is connected to a single constraint, and the mapping between the parameter value to the constraint's value is direct; for example, the parameter 'width' will hold the value of the distance in a distance constraint. However, preferably, the semantics of parameters should cover the following cases:

1. Connecting multiple constraints to the same parameter. This might be impossible using direct mapping if the constraints have different numerical values; and

2. Carrying parametric change from one feature to other features in the glyph and across the font. This might be meaningless using a direct mapping, as the same feature in different glyphs (and sometimes even in the same glyph) may have different dimensions.

To overcome these problems, a preferred embodiment of the present invention comprises a connecting mechanism that translates the parametric changes to numerical values at each constraint. This mechanism is intuitive, simple and highly efficient. All parameters are used as multipliers for the connected constraint's values. For example, if an x distance constraint with value d is connected to a width parameter w then the new distance used in the constraint will be $d \times w$. This implies that all parameters are initially set to 1.0, and may vary between 0 and infinity. Parameter values between 0 and 1 decrease the size of the constraint value, and parameter values greater than 1 increase it. Restricting the values to be positive also has the advantage of preserving the sign of the parameter, consequently preserving the order of points.

Constraints with a zero numeric value cannot be linked to a parameter, as changes in the parameter would not effect the constraint at all. However, it is possible for such constraints to be translated to some other constraint having non-zero value, for example, by using additional support points.

Different constraints having different values can now be connected to the same parameter, since they are only multiplied by it. On the other hand, the link between them is intuitive as they all decrease or increase at the same rate. This is also true for carrying parametric change across the font. Features having different dimensions only multiply their sizes by the same parameter and change in the same intuitive manner. A constraint value can also be overridden with new values by setting it to be the product of the value and a parameter (e.g. $d = d \times w$).

Using this mechanism, parameters effects are not limited to a single constraint or a single geometric entity. A parameter can be connected to several different constraints in different geometric entities, enabling global control over shapes using a single parameter shown in Figs. 6A-6C to which reference is now made.

By connecting one parameter such as "beta Round" (fig. 6A) to several different constraints (both X and Y ratio of different ratios) global change of the shape can be made. For example, Fig. 6B illustrates the letter "O" having both X and Y ratio constraints connected to the parameter "beta Round", which has a value of 1. By altering the parameter value to 2.3, the shape shown in Fig. 6C is formed. Similarly Fig. 6D is formed by altering the value of "beta Round" to 3.8.

Moreover, parameter change is not limited to one feature or glyph. A parametric change can be carried from one feature to other features in the glyph and across to other glyphs in the font, as shown in Figs. 7A and 7B. Fig. 7A illustrates the letter "M" where the increase in the width of the middle stem is carried over to the other stems. Fig. 7B illustrates the carrying over of the increased width of the middle stem to other glyphs (the associated letters N, O and P).

Some complex variations can be achieved only by modifying several parameters simultaneously. For example, changing the weight of a glyph involves modifying several different parameters in different features, and in different rates. In order to present a higher level interface for the user, parameters can be grouped together to define an axis parameter. Axis parameters are stored in a table that converts the unit of change of the axis parameter to the appropriate units in the other parameters. If a is the axis value in the range $-R \le a \le R$ then for each parameter p included in this axis, we define F_p to be the parameter factor, S_p to be the parameter shift, then the map from a to p is:

$$p = F_p^{\frac{(a+S_p)}{R}}$$

Figs. 8A-8D show the generation of bolding axis for the `B' glyph. It is implemented by grouping a change in "width" attribute of all features such as stems and bows (Fig. 8B). Then, a serif "length" parameter is added to correct serif shapes, (Fig 8C). Finally the glyph intersects the baseline: a bar "shift" parameter is added and used for translating the lower bar, namely the distance of the bar guiding line from the base line (Fig. 8D).

Constraints are defined between the geometric entities representing a feature. They allow incorporating relations between geometric entities into the glyph, and defining desired behaviours of shape changes according to parameter modifications. Such modifications trigger the constraint solver to be executed and compute a new glyph geometry.

There are several existing constraint solving techniques some of them listed in Bouma W. Fudos, I.: Hoffmann. C.M, Cai, J., Paige. R. in Geometric constraint solver. Computer-Aided Design, Vol. 27. No. 6. 1995, 487-501. Numerical solvers (such as, Newton-Raphson) are not efficient and require a good initial guess to converge and symbolic solvers are too slow. Graph-based solvers explicitly represent the system as a graph, and use it to solve the system. The graph nodes represent geometric entities and the arcs represent the constraints between them. The preferred method is graph-based method that transforms the original graph to a directed acyclic graph (DAG), such as described

by Owen, J.C.: in Algebraic Solution for Geometry from Dimensional Constraints. ACM/Siggraph Symp. On Solid Modeling and Applications, Austin. June 1991, Rossignac. J Turner, I, (eds). ACM Press, 397-408., and by Bouma W. Fudos, I.: Hoffmann. C.M, Cai, J., Paige. R. in Geometric Constraint Solver. Computer-Aided Design, Vol. 27. No. 6. 1995, 487-501. The DAG can be viewed as a procedural `program' to solve the system.

The present invention, by taking into account the special characteristics of the domain of typography, gives expressive power to designers or animators using the fonts. The complex behaviours of elements are also incorporated in the design, especially when a change in the topology of the glyph is needed (as described hereinbelow). In addition, LiveType constraints are linear, guaranteeing a single solution, and the LiveType system of constraints is a special type of DAG, enabling an efficient, procedural evaluation.

Constraints in the model of the present application determine the position of points by using zero, one or two constraints for each point. The constraint types chosen yield only a single possible solution for the point position. This means that the system does not need any complex or heuristic mechanism to select between different possible solutions. The constraints are algebraic, hence can eventually be translated to linear equations on the glyph point coordinates.

Reference is now made to Figs. 9A-9C. An unconstrained point possesses two degrees of freedom (DOFs) Fig. 9A. A point possessing a single linear DOF can move along a line called the line of freedom (LOF) illustrated by arrows in Fig. 9B. A fully constrained point possesses no DOFs (Fig. 9C). All types of LiveType constraints can be formulated as giving a LOF. The position of a fully constrained point is determined by intersecting its two LOFs.

Most of the LiveType constraints distinguish between the x and y coordinate of a point. That is because the vertical and horizontal directions possess an inherent significance in typography. Letters and text in all languages are mostly placed above, below or along straight lines. The letters themselves are displayed perpendicularly to the guiding line. Much of the feel of a certain

typographic design comes from the frequency of display arranged along lines. Many font variations are performed along one of these directions. For example, condensing and extending a font design is accomplished by horizontal scale of distances inside the glyphs (note that this scaling is not a simple affine scaling). Likewise, most grid fitting procedures during rasterization use separate horizontal and vertical passes in order to arrange elements along grid lines, and use separate vertical and horizontal interpolations for smoothing the shape of glyphs. The constraints defined in the present application are between points and points or between points and lines (for example, as shown in Figs. 3A and 3B). The following constraint types (illustrated in Figs. 10A-10C) which are be powerful enough to create meaningful and interesting shape variations are used:

1. XAbs, YAbs: point $p_0 = (x_0, y_0)$ must retain its x or y coordinate. The lines of freedom are $x = x_0, y = y_0$.

2. XDist, YDist: the x or y coordinate of point p_0 must remain in signed distance d from the x or y coordinate of point p_1. The lines of freedom are $x = x_1 - d, y = y_1 - d$ (Fig. 10A). Note that since the distance d is signed, there is only a single line of freedom.

3. LineDist: point p_0 must remain in signed distance d from the line passing through points p_1 and p_2. If $ax + by + c$ is the line equation and $R = \sqrt{a^2 + b^2}$, then the line of freedom is $ax + by + c - Rd$ (Fig. 10B).

4. Angle: point p_0 must lie on a line having angle α with a line defined by p_1, p_2 and passing through p_1. The line of freedom is obtained by rotating p_0 around p_1.

5. XRatio, Yratio: the x or y coordinate of point p_0 must remain in ratio t from the x or y coordinates of points p_1 and p_2. The lines of freedom are $x = x_1(1-t) + x_2 t, y = y_1(1-t) + y_2 t$ (Fig. 10C).

An example for defining constraints in a bar feature and the part of the constraint graph it imposes can be seen in Figs. 11A and 11B. Fig. 11A illustrates the bar feature of the letter 'A' and Fig. 11B shows the sub-graph of constraints it imposes. Increasing the width parameter will increase the distance between

points 2,3 and point 1, which in turn translates points 5, 6, 7, 8 preserving the bar's shape. Additional constraints can be added to the same glyph in order to define different variations. Naturally, there is a limit to the number of variations that can be supported by the same underlying geometry because we require the system to be well-constrained.

The primary task of glyphs is to be displayed rapidly. This means that the LiveType constraint scheme must guarantee finding a solution at a predictable rapid time. Preferably, only systems of constraints that can be transformed to a DAG and solved procedurally are used. The LiveType constraint scheme utilizes directional constraints rather than stating a symmetric relation between objects, and only constraint cycles in the graph which can be handled by the algorithms such as described by Owen, J.C.: in Algebraic Solution for Geometry from Dimensional Constraints, at the ACM/Siggraph Symp. On Solid Modeling and Applications, Austin. June 1991, Rossignac. J Turner, I, (eds). ACM Press, 397-408., and by Bouma W. Fudos, I.: Hoffmann. C.M, Cai, J., Paige. R. in Geometric Constraint Solver. Computer-Aided Design, Vol. 27. No. 6. 1995, 487-501.

It is a feature of the present application to use 'mixed-ratio graphs' which are defined as directed graphs that do not contain cycles after removing either all the Yratio constraints or all the XRatio constraints. An example of ratio constraint effect of translating a stem feature inside a glyph is illustrated in Figs 12A-12D.

The effect of applying ratio constraints is shown by the letter "O" of Fig. 12A with ratio constraints (Fig. 12C) contrasted with Fig. 12B (without ratio constraints). Ratio constraints connect each intermediate point to the extreme points of its contour (Fig. 12D), distributing shape changes gradually along the outline points, and preserving the glyph's continuous shape.

In most cases, the XRatio and YRatio constraints are defined by relating a point inside a contour to the two closest local extreme points in the same contour (Fig. 12D). These constraints are responsible for smoothing the shape of glyphs during dynamic variations (Fig. 12A). They are used as default constraints

whenever an outline point does not possess another constraint in the x or y direction respectively. For that reason, ratio constraints are the most prevalent ones. The difficulty is that even in the most simple shapes, using both x and y ratio constraints will create cycles in the constraints graph (see Figs. 13A-13B). Therefore, in order to use ratio constraints, mixed ratio graphs were defined to overcome this difficulty and Applicants have devised an algorithm that is capable of evaluating cyclic graphs. The evaluation algorithm, which can handle mixed ratio graphs using a two step pass is described hereinbelow (see Fig. 15A).

Fig. 13A illustrates a simple circle shape and Fig. 13B illustrates the mixed ratio constraint graph imposed by the shape.

Fig. 14A illustrates the underlying geometry of the letter 'A' together with a portion of its mixed ratio graph. Each point has not more than four outgoing arcs pointing to its constraining points, but might have several incoming arcs coming from points constrained by it.

Reference is now made to Figs. 15A and 15B. Fig. 15B is a high level flow chart illustration of the method for re-evaluating the mixed ratio constraints graph whenever a parameter is altered, as described by the pseudo source code of Fig. 15A.

At the first stage, all points in the glyph are marked as unevaluated, (step 202) and all YRatio constraints connected to outline points are removed from the constraint graph (step 204). Looping on all points in the glyph, a recursive evaluation procedure for each point is called (step 206). At the second stage, all points in the glyph are again marked as unevaluated (step 208), all XRatio constraints connected to outline points are removed (step 210), and the same loop is carried out again (step 212).

The evaluation of a single point is illustrated by the high level flow diagram (Fig. 15C), to which reference is now made. If a point is marked as evaluated (query box 214), it returns its current position (step 216). If the point is unevaluated, its position is calculated based on its constraints. Constraints can be expressed as yielding lines of freedom (LOF). Each LOF is computed, based

on the constraining points, which are evaluated recursively by the same procedure.

The constraints (c1 and c2, say) for each point are obtained (step 220). If both c1 and c2 are empty (query box 222), it returns its current position (step 216). A check is then made to see whether either c1 or c2 is empty.

If c1 is empty (query box 224), a check is made whether c2 is an X constraint (query box 226). If c2 is an X constraint, then c1 = Yabs (228). Otherwise c1 = Xabs (230). Then a check is made to see whether c2 is empty.

If c2 is empty (query box 232), a check is made whether c1 is an X constraint (query box 234). If c1 is an X constraint, then c2 = Yabs (236). Otherwise c2 = Xabs (238).

Following these steps (234 - 238) and in the case where c2 is not empty, the lines of freedom (LOF) L1 and L2 for c1 and c2, respectively, are computed (step 240) and the current positions of the point is set to the intersection of these lines. The current position is then returned (step 216).

Some points in the graph might be under-constrained. Specifically, this refers to the leaves of the graph or points that include XRatio or YRatio constraints that were removed by the first or second evaluation step respectively. In order to make the constraint system a well-constrained one, the current positions of such points is taken as a default constraint using XAbs or YAbs to fix their location in the x or y direction.

The algorithm executes several times consecutively, switching between primary x and primary y directions in a continuous manner, so that the result is effectively order independent.

Let k be some number representing the longest time for calculating two lines of freedom from their geometries and finding their intersection. Such calculations are done $2kn$ times for each parametric change, where n is the number of glyph points (outline and supporting). A point is dependent on at most four other points. Therefore, evaluation time for a single pass is bounded by $4n+2kn$ (this happens if all points were constrained, which is impossible because there must be leaves to the DAG). Therefore, evaluation time is linear in the

number of points, with a small constant. Note that the time it takes to re-compute the glyph's geometry is at least as efficient as the time it takes to rasterize a glyph.

There are cases when a change in a parametric value might lead to undesirable results involving a change in the topology of the shape, such as a change in the order of points and introduction of new intersections as illustrated in Figs. 16A - 16E, to which reference is now made.

Fig. 16E illustrates the Courier Font letter 'M' and the effect of bolding caused by increasing the width parameter of the line distance is shown in Figs. 16A-16C. The constraints cause an undesirable topological change in shape, as shown by Fig. 16C where points 1 and 2 are pushed away from lines a and c, leaving a triangular "hole". Fig. 16D illustrates the relationship between points 1 and 2 and lines a, b and c.

The state of a glyph's constraints system can be fully determined by the set of glyph points *P* and the set of constraints defined on these points *C(P)*. As described hereinbelow, some constraints are connected to external parameters, and therefore a fixed set of constraints can produce several glyph shapes by altering parametric values. We define a set *S* of states *S = 0, 1, ..., r* such that for each $i \in S$, $C_i(P)$ is a set of constraints defined on the given set of points P. The constraints $C_i(P)$ of the states i impose different mixed ratio graphs. Therefore, we can evaluate the constraint set $C_i(P)$ using our evaluation algorithm, resulting in a glyph shape.

A glyph state $i \in S$ is implemented in our model as the union of all states i_p for all $p \in P$. Each point has several constraint states, and the state of the glyph is the union of all of its point states. Each point has a default start state, including its default constraints.

States are used by defining a set of transition rules that switch from using one concrete constraints system to another. There are many types of transition rules possible, including outside events or user interaction.

The preferred rules are specified by a simple Boolean expression evaluator whose variables are geometric properties of points. These can include any expression of the following form:

$$\exp = \exp \vee \exp$$
$$\exp = \exp \wedge \exp$$
$$\exp = \sim \exp$$
$$\exp = coordinate\ op\ coordinate$$
$$coordinate = pnt.x \mid pnt.y$$
$$pnt = 0 \mid 1 \mid 2 \mid 3 \ldots$$
$$op = < \mid > \mid = \mid \neq \mid \leq \mid \geq$$

In practice, states are specified by the font designer through a graphical user interface. The designer visually recognizes a situation in which the constraints governing the behavior of a glyph point should be modified. The geometric properties of this situation are specified for example that two glyph points coincide into one, as shown in Figs. 17A-17F, and are encoded internally by the system as transition rules represented by Boolean expressions. The designer supplies the constraints for the new state. The font user is not required to be aware of constraints and states.

Figs. 17A-17F show the transition rule and the portion of constraints graph which changes in two states (Figs. 17D and 17F) to correct the behavior of the Courier 'M' points while bolding axis is used. Constraint states are used to overcome topology changes during dynamic parameterization.

Figs. 17A -F illustrate the use of constraint states to solve topological problems. An additional point 3 is added (Fig. 17A). A transition rule (fig. 17E) using this point between two constraint cases (Figs. 17D and 17F) is defined to govern the movement of points 1 and 2. The change due to bolding is illustrated by Figs 17A- 17C. Fig. 17B illustrates an intermediate position between the non-bold Fig.17A and Fig. 17C. Points 1, 2 and 3 remain together during bolding. It will be appreciated by persons knowledgeable in the art that the use of constraint states allows for varying the degree of change of a bolding or other characteristic.

Global system states have been used in the constraint framework presented by Kalra,. D, Barr, A.H. in A Unified Framework for Constraint-Based Modeling, in the proceedings, Computer Graphics International '93. Springer-Verlag. 1993, pp. 675-695. Transition rules are general expressions and are usually associated with geometric configurations. Freeman-Benson. B.N., Borning, Al, in Constraint Imperative Programming Languages for Building Interactive Systems, in Languages for Developing User Interfaces. Brad. A. Myers (Ed.). Jones and Banlett, 1992, pp. 161-183. use system states in the context of integrating constraints with an imperative programming language for implementing interactive systems. The states used are not based on geometry, but are used for defining a new programming language paradigm. In the present application each low-level geometric entity possesses its own state. In addition, the transition rules in by Kalra,. D, Barr, A.H. in A Unified Framework for Constraint-Based Modeling, in: proceedings, Computer Graphics International '93, Springer-Verlag. 1993, pp. 675-695 are all associated with time, because the application refer to animation.

It will be appreciated by persons skilled in the art that the present invention is not limited to a single glyph but is also applicable to a font model. The single glyph described hereinabove is self contained in that all the underlying geometry, features and constraints defining a glyph are stores as a single unit. This self-containment has the advantage that in order to see the glyph, it is not necessary to see the complete font. Thus, a "compressed" font model can be constructed in which features which are common to several glyphs are stored only once.

The compressed font model, generally designated 100, which is illustrated with reference to Fig. 25, to which reference is now made, is obtained by several simple modifications to the LiveType design glyph model 10, described hereinabove with respect to Fig. 1A. Elements of this embodiment of the invention which are similar to elements which have been previously described with respect to the preferred embodiment hereinabove, are similarly designated and will not be further described.

The components of the basic glyph model 10 are its underlying geometry 12 and its feature hierarchy 14. The feature hierarchy 14 comprises a plurality of features 102. The main components comprising a feature 102 (similar to 15 in Fig.1B) are its geometry (boundary and support points) 20, constraints 22 and parameters 24. The features of all the glyphs forming the font model 100 are stored in a feature store 104.

The modifications can be summarized as follows. First, the glyph model does not store its underlying geometry; instead, each feature stores its own geometry. Secondly, the glyph model does not store its features; it only points to a 'feature store' common for all glyphs in the font. Thirdly, the feature hierarchy inside the glyph is enhanced by attaching an affine transformation to every pointer to a child feature (102). This transformation should be applied on the feature geometry to bring it to the same coordinate system as its parent in the feature hierarchy. A glyph in the compressed font model it thus emptied of some of its content; essentially, the main remaining component is the feature hierarchy, enhanced by the affine transformations.

Regarding features, there are three main modifications to the design glyph model features. First, features are now stored in a global, font level location, rather than inside every glyph. Second, each feature stores its own underlying geometry (though actually, features could still share geometry). Third, each segment in a feature's geometry is optionally labeled either as an outline segment or an auxiliary segment. This optional labeling can be used when composing the outline of a glyph from its features, as described below. In addition to these changes, parameter signatures may be arranged either globally or locally in a feature, depending on their intended usage and type.

When there is a need to display a glyph or to compute its outline for some other reason, the following modifications are made to the evaluation algorithm (Figs 15A-15C). For every feature, its final affine transformation is computed (it is simply the concatenation of the affine transformations leading to it from the root feature of the glyph). Now the algorithm described in Figs. 15A- 15C is invoked, resulting in the feature's geometry.

There are two ways to compose the geometries of different features. First, the segments (lines or curves) originating from all the features can be connected together, removing the segments denoted as auxiliary segments and retaining only the outline segments. Because the glyph was designed properly, the outline segments belonging to different features automatically form a valid glyph outline. A second alternative is useful for rasterization purposes. During design, each feature is decomposed into a solid area that covers part of the glyph and into other auxiliary parts. After the affine transformation and the evaluation process, all solid areas of all features are rasterized independently, producing a bitmap of the whole glyph. Of course, rasterizing hints should still be used for high-quality rasterization in both options.

The compressed font model results in glyphs that are composed of features that are identical up to an affine transformation. Alternatively, It is possible to modify parameters of the same feature differently in different glyphs, or even to modify the geometry itself of a feature in some glyph. For this purpose, in addition to the attached affine transformation, a pointer feature in a glyph can store other modifications to be performed on the feature after it is evaluated in the glyph and before it is used for composing the final glyph bitmap. For example, points representing serifs can be moved finely in order to create fine differentiations between different instances of the same general serif.

Examples

Figs. 18-21 show several illustrations for possible applications of the LiveType glyph . Figs. 18A-D show several snapshots from dynamic behaviours of letters: in Figs. 18A and 18B, classical typographic font family axes, weight and slant, were defined as LiveType axes for the fonts Times (a) and Courier (b), and were used to create dynamic modifications of letters. Fig. 18C shows the effect of changing the shape of serifs (the snow flakes were added externally). In Fig. 18D, letters from different fonts were extruded in a 3-D environment, while retaining their typographic nature as letters and being manipulated accordingly.

Figures 19–21 show some additional dynamic behaviours, which are used either to enhance and illustrate the semantics of the word or for pure fun.

To illustrate the variations which may be made to existing font models using the LiveType glyph of the invention, reference is now made to Figs. 22-24. Fig. 22 illustrates the glyphs of characters A, B and C from the Courier new font. The characters A, B, C contain several different features, such as stems, bows, arcs, bars, serifs, counters and joints.

Figs. 23A and 23B illustrate variations stemming from a single 'A' model along three different axes: horizontally, slanted variations; vertically, weight variations; and width variations between the two figures. Slant and weight axes of 'B' and 'C' are shown in Fig. 24. The glyphs bounded by squares are similar to the limited variations provided by current outline representations.

As opposed to interpolation methods, which provide only interpolations between two extremes, a LiveType model can support highly diverse variations by utilizing the same set of constraints with different grouping of parameters. For example, Figs. 8A-8D and Fig. 17A-17C show complex modifications of glyph shapes yielding variations that cannot be achieved by interpolation methods.

LiveType is much more space efficient than other models for font families. In addition, LiveType is much more powerful than existing models, supporting non-linear deformations without losing design intent.

In addition to the essential variations shown above, LiveType supports non-linear navigation in design space by effortless manipulation of parameters. Examples of possible variations are illustrated in Fig. 26 and 27.

In addition, the inherent variational capabilities of LiveType can be used by end-users interactively in order to create interesting font effects. These could be functionally useful in order to emphasize the user's intentions, or be done simply for aesthetic pleasure and pure fun.

It will be appreciated by persons skilled in the art that the present invention is not limited to what has been particularly shown and described hereinabove. Rather the scope of the present invention is defined only by the claims which follow.

CLAIMS

1. A glyph defined by its geometry and comprising a plurality of features each of said features being defined by at least one constraint and at least one parameter attached to at least one of said plurality of features.
2. A glyph according to claim 1 wherein each of said plurality of features are further defined by its boundary and support points.
3. A glyph according to any of the previous claims wherein said plurality of features comprises typographic features and supporting features.
4. A glyph according to claim 3 wherein said typographic features include at least one of a group including bars, stems, bows, arc, curve stems, curve bars and serifs.
5. A glyph according to any of the previous claims wherein at least two of said plurality of features are treated as a single stroke unit.
6. A glyph according to any of the previous claims wherein said at least one constraint comprises at least one of a group including distances between points and lines, ratios of said distances and angles between lines.
7. A glyph according to claim 1 wherein each of said plurality of features comprises a plurality of parameters combined together to form a typographical axis parameter whereby by changing a single axis value, at least two of said plurality of parameters are correspondingly changed.
8. A glyph according to claim 7 wherein the rate of change of one of said at least two of said plurality of parameters is independent of the rate of change of said second parameter.
9. A glyph according to claim 1 wherein a change to each of said at least one parameter is connected to at least one of said at least one constraint.
10. A glyph according to claim 9 wherein said at least one constraint has a value which is a multiplication of a scalar and said at least one parameter.

11. A glyph according to claim 1 wherein at least two of said plurality of features are connected to a single parameter, whereby by changing said single parameter said at least two of said plurality of features are correspondingly changed.

12. A font model comprising a plurality of glyphs, each of said plurality of glyphs being defined by its geometry and comprising a plurality of features wherein each of said features is defined by at least one constraint.

13. A font model according to claim 12 and wherein each of said plurality of glyphs comprises at least one parameter attached to each of said plurality of features.

14. A font model according to claim 13 wherein each of said plurality of features are further defined by its boundary and support points.

15. A font model according to any of claims 12 - 14 wherein said plurality of features comprises typographic features and supporting features.

16. A font model according to claim 15 wherein said typographic features include at least one of a group including bars, stems, bows, arc, curve stems, curve bars and serifs.

17. A font model according to any of claims 12 - 16 wherein at least two of said plurality of features are treated as a single stroke unit.

18. A font model according to any of claims 12 - 17 wherein said at least one constraint comprises at least one of a group including distances between points and lines, ratios of said distances and angles between lines.

19. A font model according to claim 12 wherein each of said plurality of features comprises a plurality of parameters combined together to form a typographical axis parameter whereby by changing a single axis value, at least two of said plurality of parameters are correspondingly changed.

20. A font model according to claim 19 wherein the rate of change of one of said at least two of said plurality of parameters is independent of the rate of change of said second parameter.

21. A font model according to claim 12 wherein a change to each of said at least one parameter is connected to at least one of said at least one constraint.

22. A font model according to claim 21 wherein said at least one constraint has a value which is a multiplication of a scalar and said at least one parameter.

23. A font model according to claim 12 wherein at least two of said plurality of features are connected to a single parameter, whereby by changing said single parameter said at least two of said plurality of features are correspondingly changed.

24. A font model according to claim 12 and further comprising a feature store for storing a plurality of glyph features wherein each glyph points to at least one of said stored glyph features

25. A font model according to claim 12 wherein at least one of said plurality of features within each of at least two different glyphs are connected to a single parameter, whereby by changing said single parameter said at least one of said plurality of features within each of at least two different glyphs are correspondingly changed.

26. A method for reevaluating a constraint graph for a glyph, said glyph having a plurality of points, each of which has first and second constraints, said graph having outline YRatio and XRatio constraints and a plurality of geometric constraints, the method comprising the steps of:

marking all of said plurality of glyph points as unevaluated;

removing said YRatio constraints connected to outline points;

determining the position of each of said plurality of glyph points by evaluating each of its geometric constraints;

marking all of said plurality of glyph points as unevaluated;

removing said XRatio constraints connected to outline points; and

determining the position of each of said plurality of glyph points by evaluating each of its geometric constraints;

27. The method of claim 26 wherein said step of evaluating comprises the steps of:

If a point is marked as evaluated, returning its current position;

If a point is marked as un-evaluated, obtaining said first and second constraints;

calculating said first (c1) and second (c2) constraints;

computing the lines of freedom (LOF) for each of said first and second constraints;

If both said first and second constraints are empty, returning the current position;

If said first constraint is empty and if said second constraint is an X constraint, then c1 = Yabs; otherwise c1 = Xabs;

If said second constraint is empty and if said first constraint is an X constraint, then c2 = Yabs; otherwise c2 = Xabs; and

setting the new position of the point to the intersection of the LOF of the first and second constraint.

28. A method for creating a variation of a glyph having at least one constraint, the method comprising the step of altering at least one parameter attached to said at least one constraint.

29. A method according to claim 28 and further comprising the step of reevaluating the constraints graph of said glyph to modify the geometry said glyph.

30. A method according to claim 29 wherein said step of reevaluating a comprising the steps of:

marking the plurality of glyph points associated with said glyph as unevaluated;

removing said YRatio constraints connected to outline points;
evaluating each of said plurality of glyph points;
marking the plurality of glyph points associated with said glyph as unevaluated;
removing said XRatio constraints connected to outline points; and
re-evaluating each of said plurality of glyph points.

31. The method of claim 30 wherein said step of evaluating comprises the steps of:

If a point is marked as evaluated, returning its current position;
If a point is marked as un-evaluated, obtaining said first and second constraints;
calculating said first (c1) and second (c2) constraints;
computing the lines of freedom (LOF) for each of said first and second constraints.
;If both said first and second constraints are empty, returning the current position;
If said first constraint is empty and if said second constraint is an X constraint, then c1 = Yabs; otherwise c1 = Xabs;
If said second constraint is empty and if said first constraint is an X constraint, then c2 = Yabs; otherwise c2 = Xabs; and
setting the new position of the point to the intersection of the LOF of the first and second constraint.

32. A system of constraint states for a geometric object comprising a plurality of states wherein each of said states has at least one constraint attached thereto and wherein a change from one of said plurality of states to another is determined by transition rules.

33. A system according to claim 32 wherein said geometric object is a glyph.

34. A system according to any of claims 32 -33, wherein said transition rules comprise any of a group of rules including geometric conditions or time.

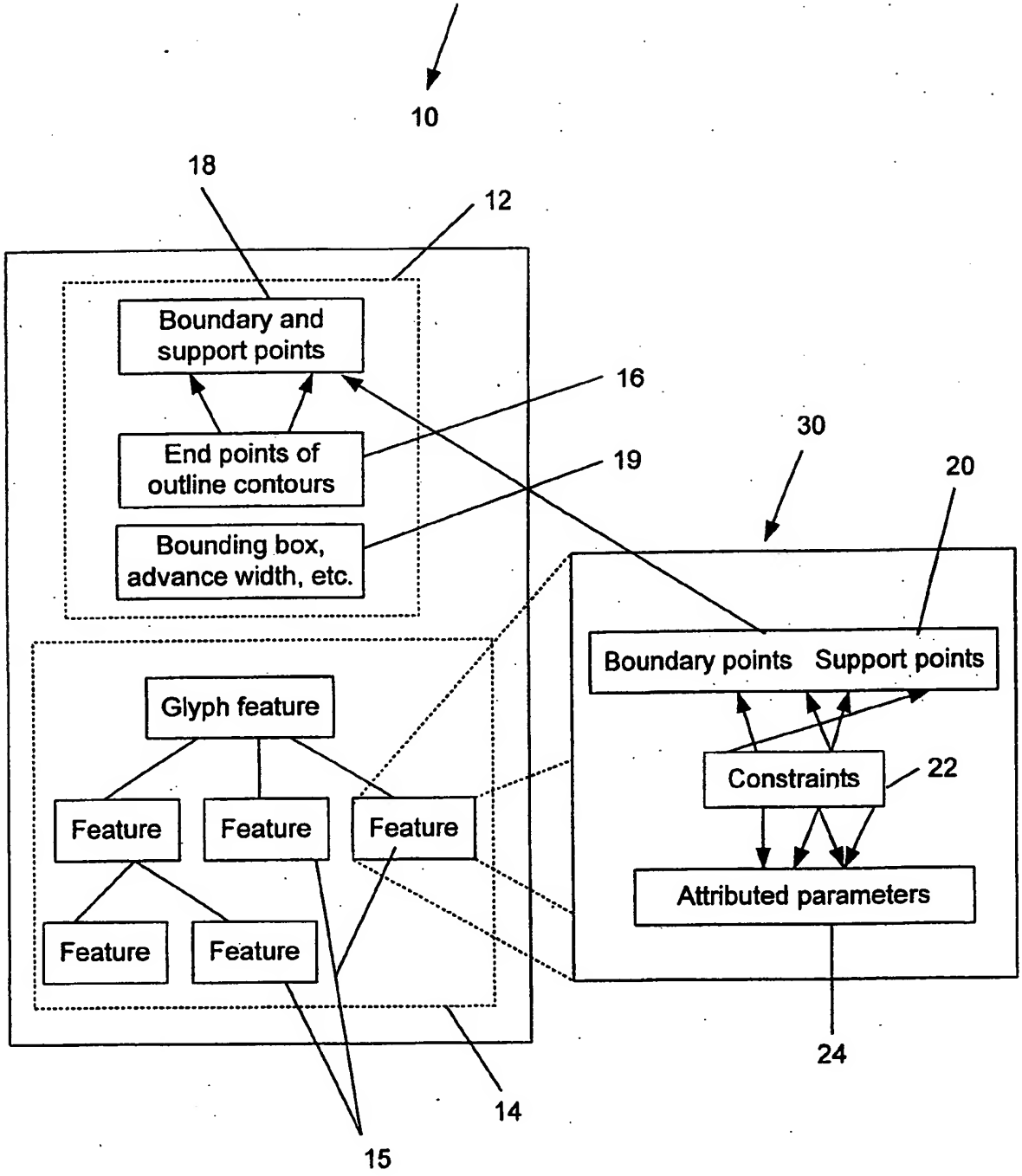

10
18
12
Boundary and support points
16
End points of outline contours
19
Bounding box, advance width, etc.
30
20
Boundary points
Support points
Glyph feature
Constraints
22
Feature
Feature
Feature
Feature
Feature
Attributed parameters
24
14
15

FIG. 1

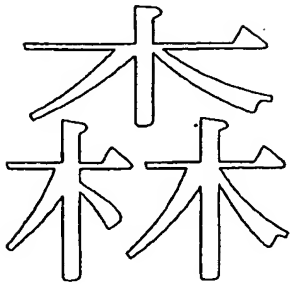


FIG. 2A

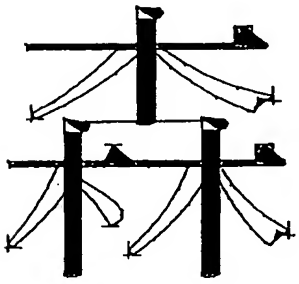


FIG. 2B

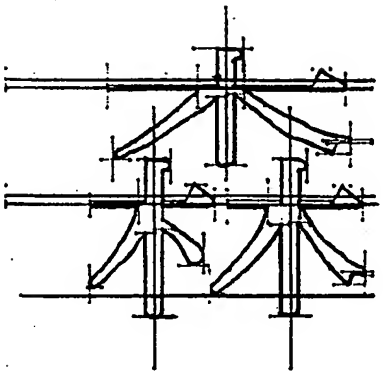


FIG. 2C

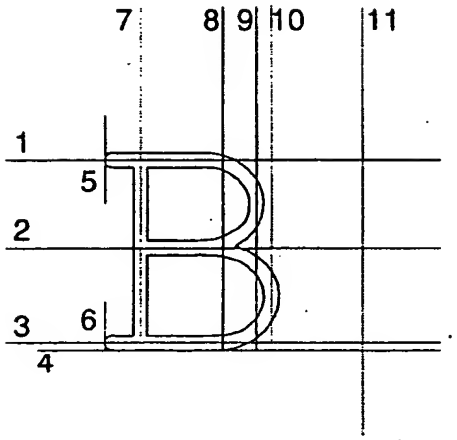


FIG. 3A

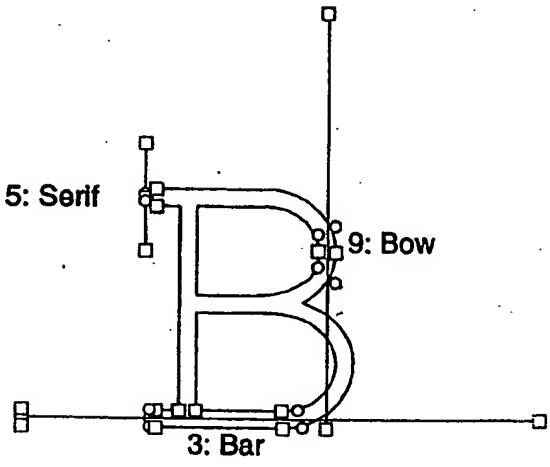


FIG. 3B

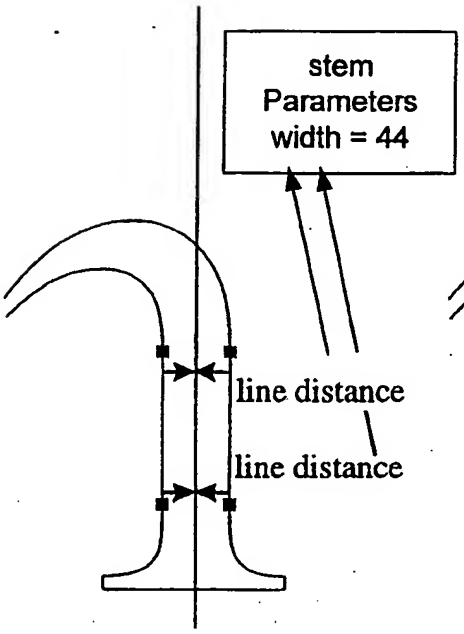

stem
Parameters
width = 44
line distance
line distance

FIG. 4A

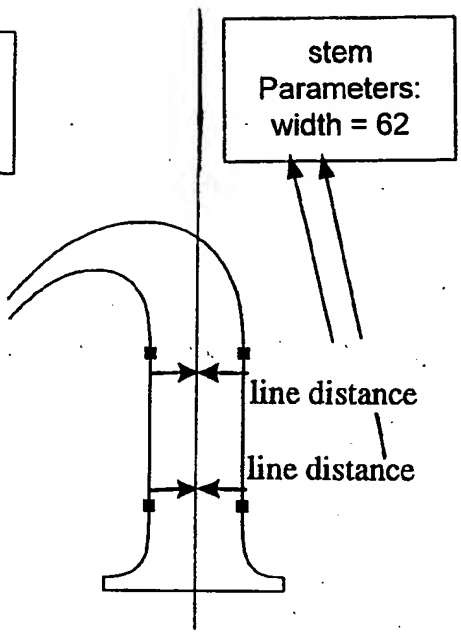

stem
Parameters:
width = 62
line distance
line distance

FIG. 4B

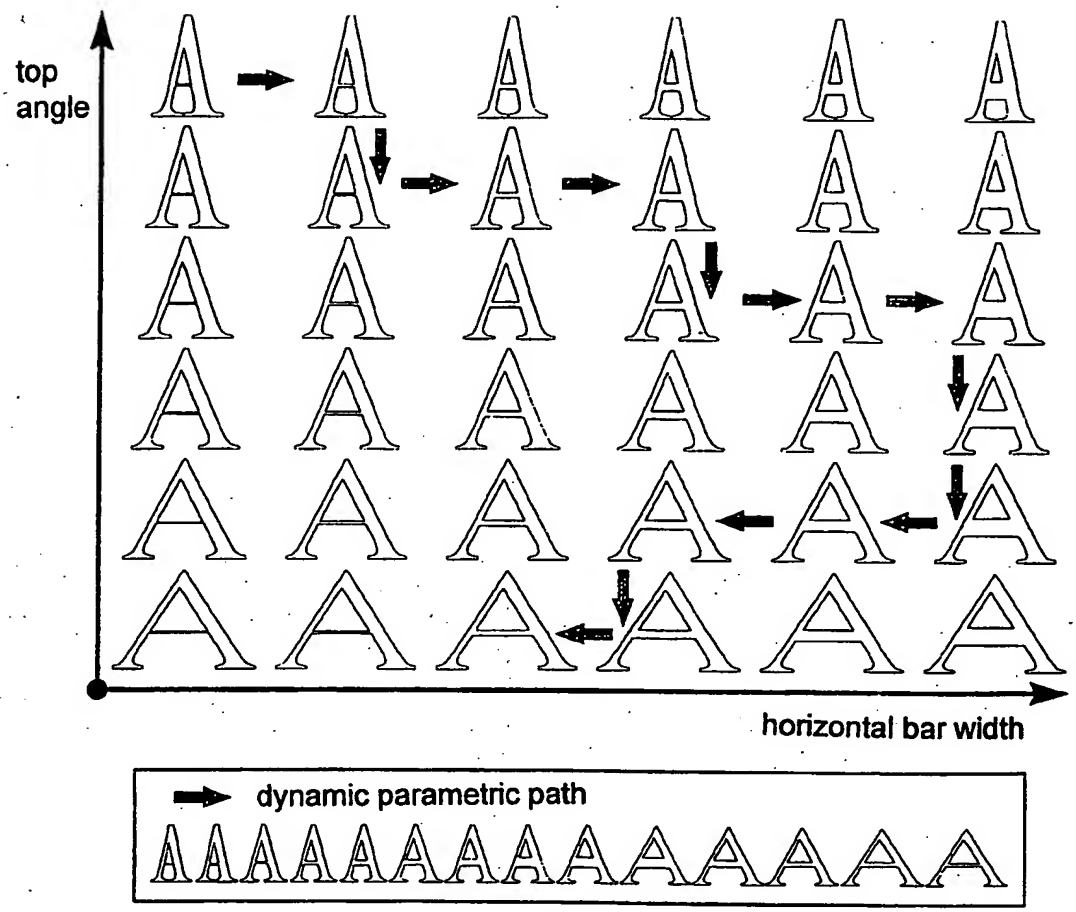


FIG. 5

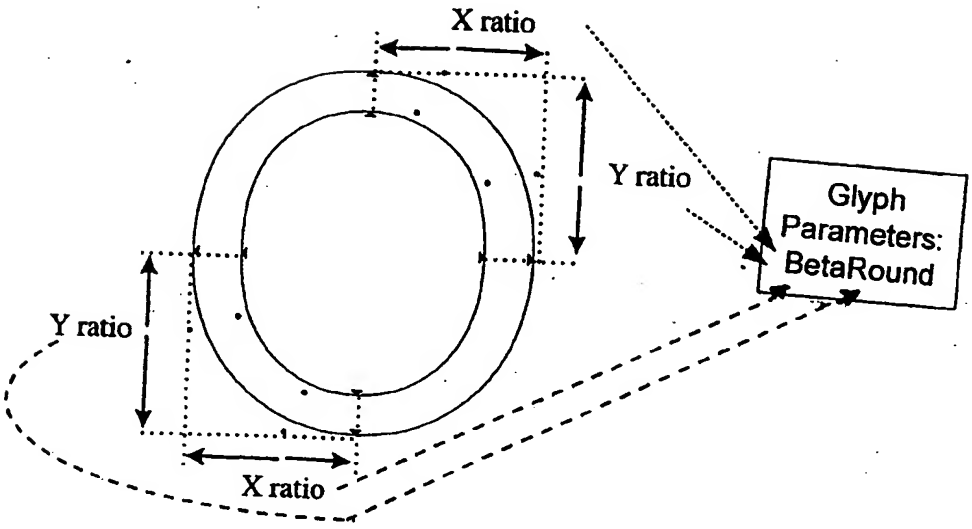


FIG. 6A

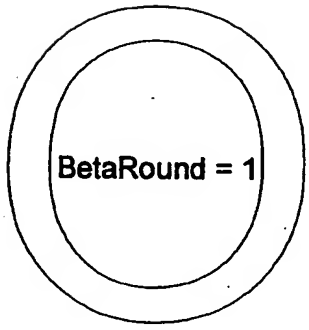


FIG. 6B

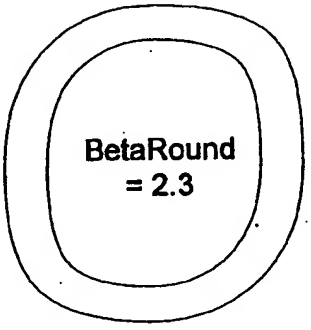


FIG. 6C

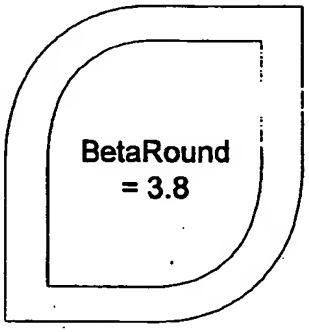


FIG. 6D

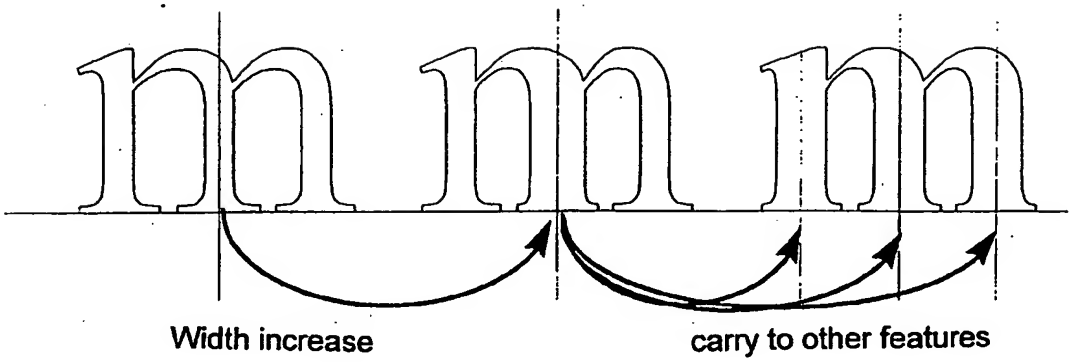

Width increase
carry to other features

FIG. 7A

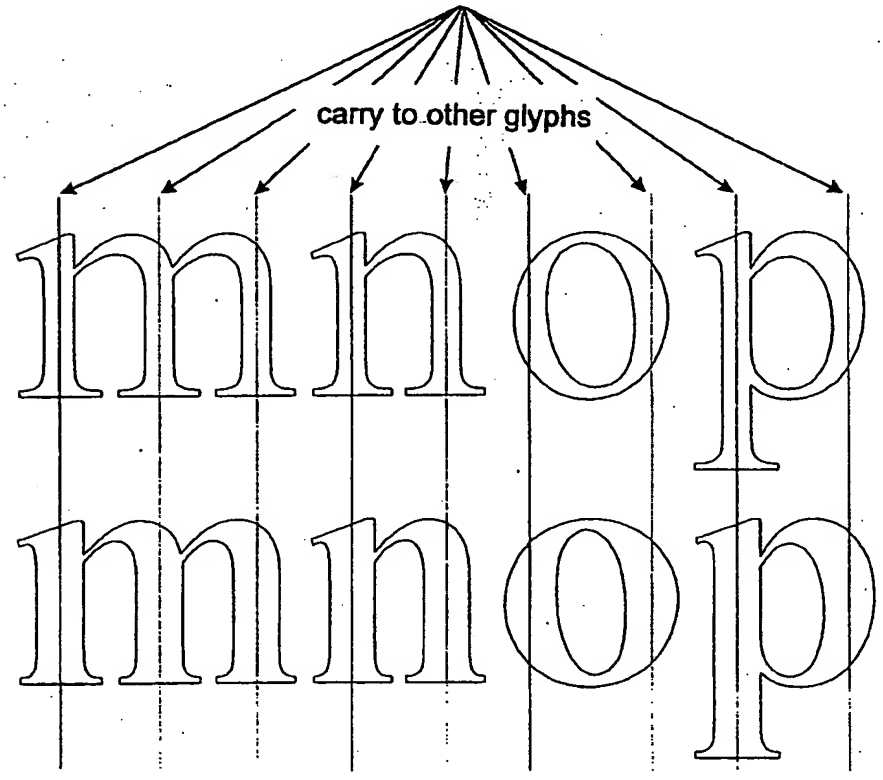

carry to other glyphs

FIG. 7B

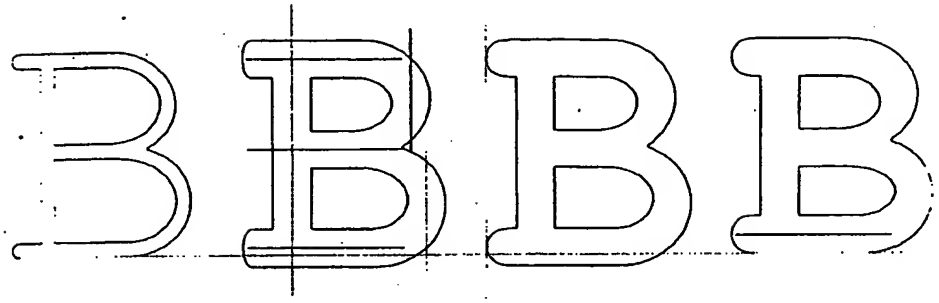


FIG. 8A FIG. 8B FIG. 8C FIG. 8D

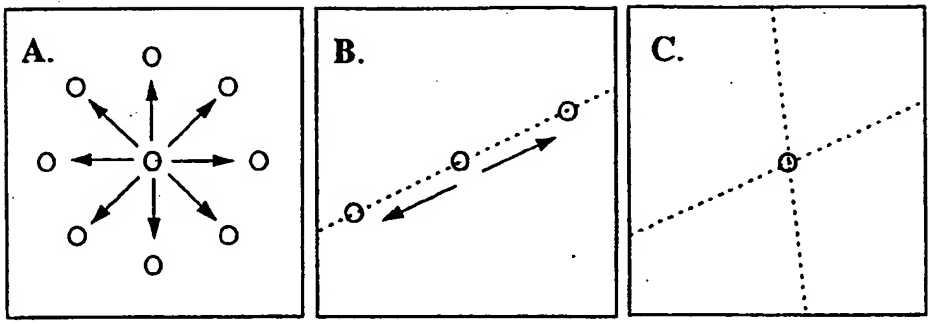


FIG. 9A FIG. 9B FIG. 9C

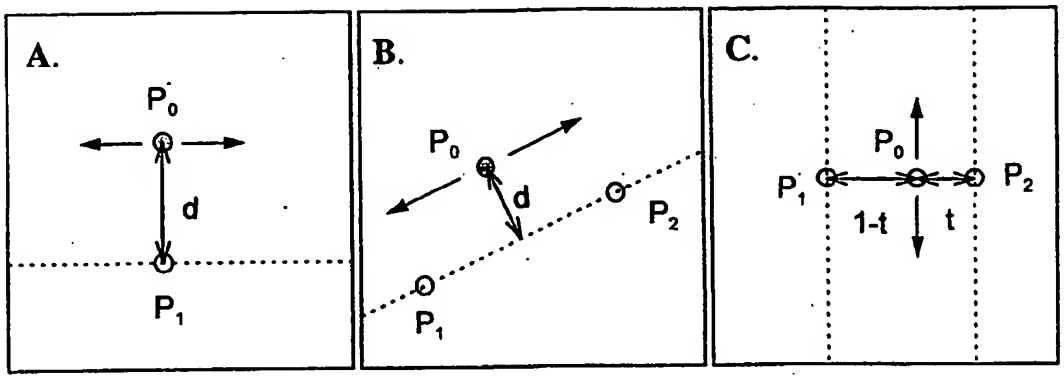


FIG. 10A FIG. 10B FIG. 10C

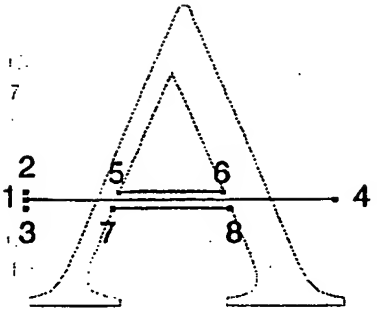


FIG. 11A

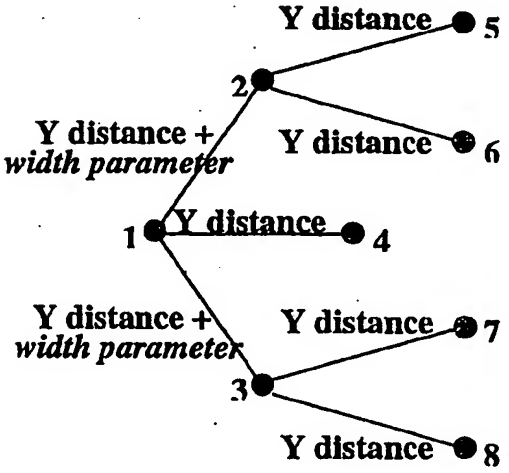


FIG. 11B

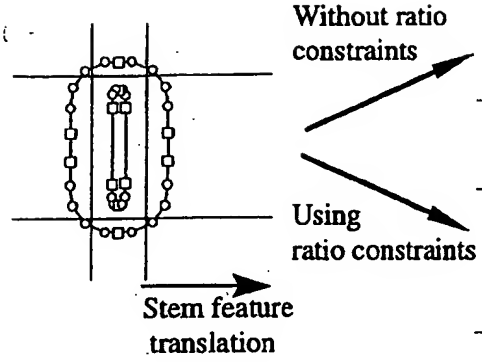


FIG. 12A

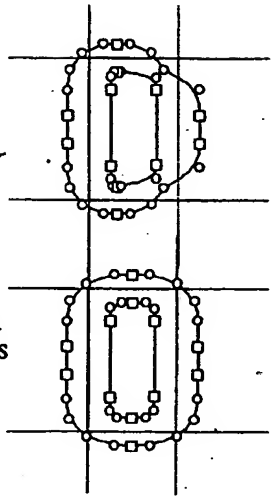


FIG. 12B

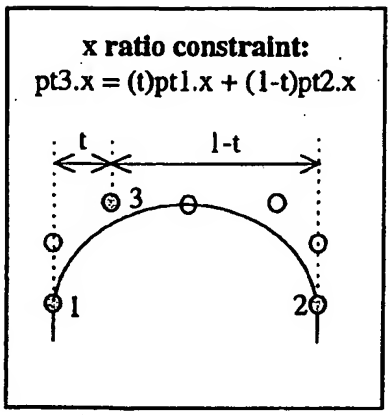


FIG. 12C

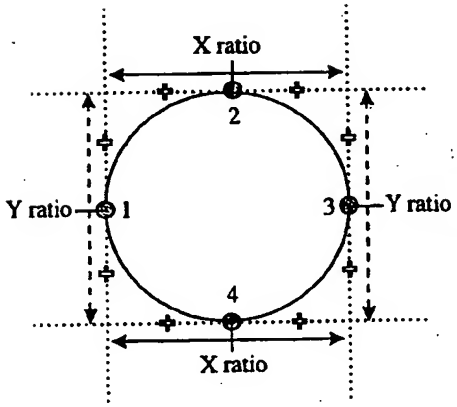


FIG. 13A

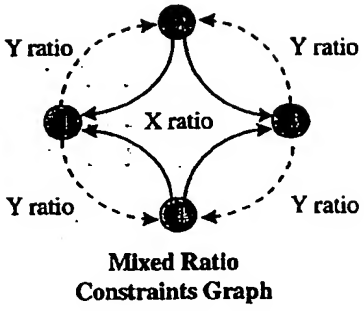


FIG. 13B

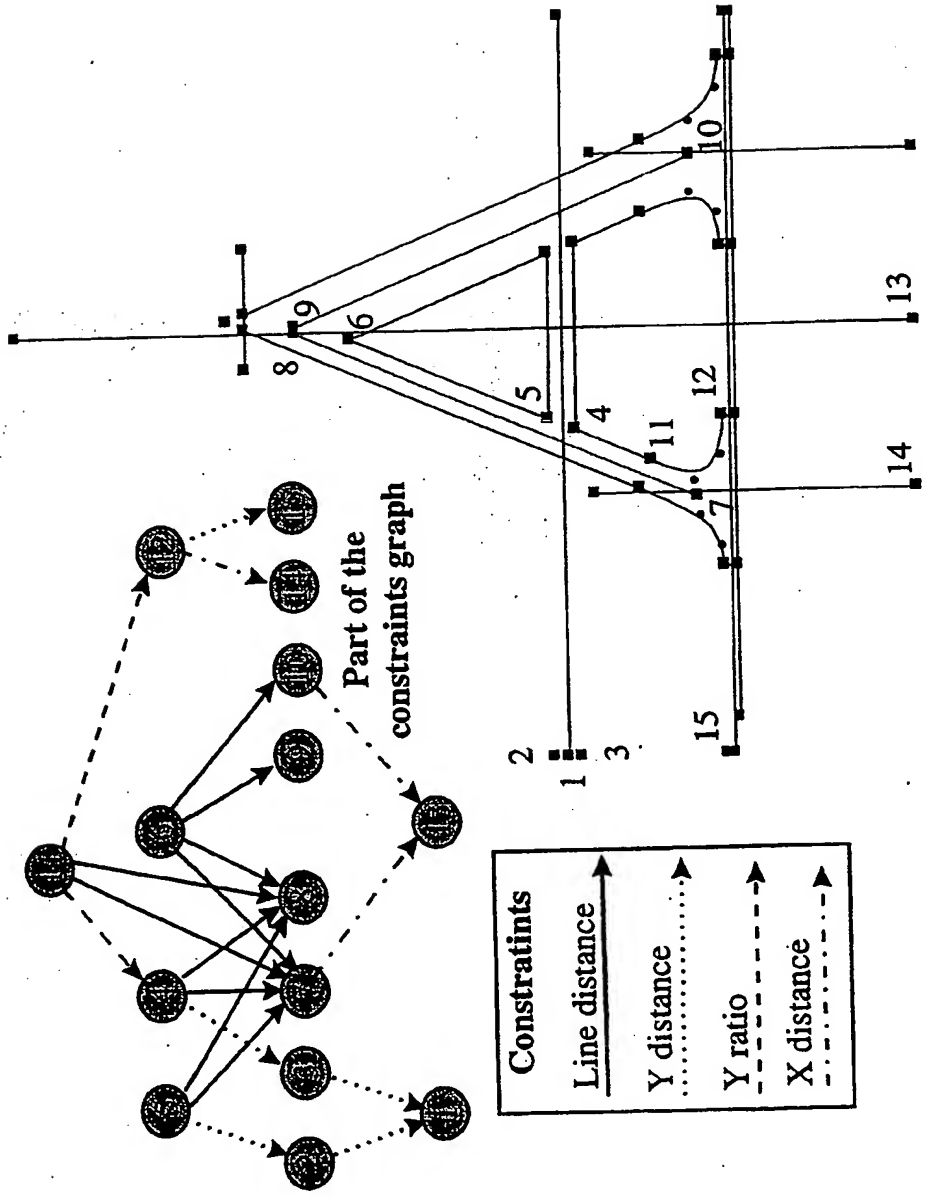

Part of the constraints graph
Constratints
Line distance
Y distance
Y ratio
X distance
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

FIG. 14

```
Glyph::evaluate ()
  G <-- this.constraintGraph
  for all points p in glyph
    mark p as unevaluated
  deactivate all outline YRatio constraints in G
  for all points p in glyph
    p. evalPoint ()
  for all points p in glyph
    mark p as unevaluatedm
  deactivate all outline XRatio constraints in G
  for all points p in glyph
    p. evalPoint ()

Point::evalPoint ()
  if (evaluated)
    return this
   c1 <-- first constraint
   c2 <-- second constraint
  if (c1 and c2 are empty)
    return this
  if (c1 is empty)
    if (c2 is an x constraint) c1 <-- YAbs constraint
    else c1 <-- XAbs constraint
  if (c2 is empty)
    if (c1 is an x constraint) c2 <-- YAbs constraint
    else c2 <-- XAbs constraint

   L1 <-- c1. getLOF ()
   L2 <-- c2. getLOF ()
  position <-- intersect (L1, L2)
  mark as evaluated
  return this

Constraint::getLOF ()
   p1 <-- first constraining point
   p2 <-- second constraining point
  return  createLine (p1.evalPoint(), p2.evalPoint())
```

FIG. 15A

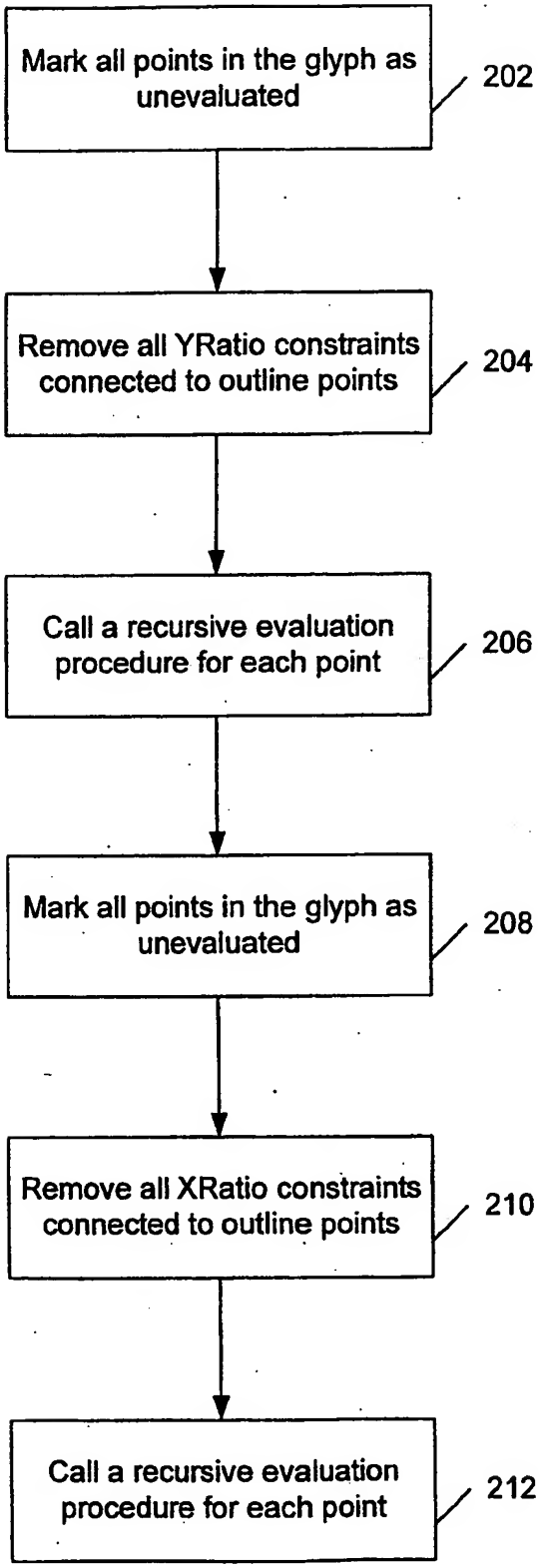

Mark all points in the glyph as unevaluated
202
Remove all YRatio constraints connected to outline points
204
Call a recursive evaluation procedure for each point
206
Mark all points in the glyph as unevaluated
208
Remove all XRatio constraints connected to outline points
210
Call a recursive evaluation procedure for each point
212

FIG. 15B

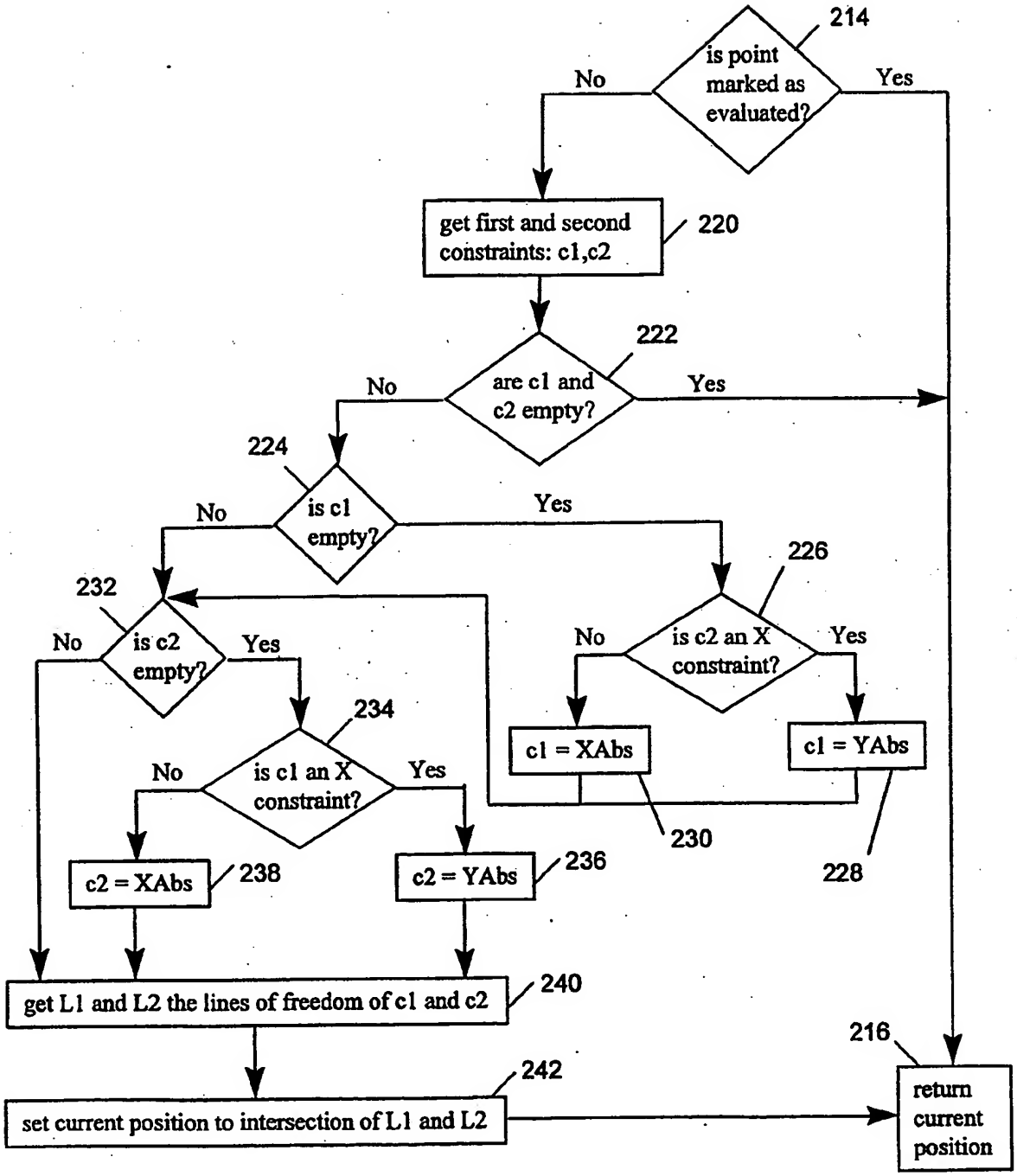

214
is point marked as evaluated?
No
Yes
get first and second constraints: c1,c2
220
222
are c1 and c2 empty?
No
Yes
224
is c1 empty?
No
Yes
226
232
is c2 empty?
No
Yes
is c2 an X constraint?
No
Yes
234
is c1 an X constraint?
No
Yes
c1 = XAbs
c1 = YAbs
230
228
c2 = XAbs
238
c2 = YAbs
236
get L1 and L2 the lines of freedom of c1 and c2
240
216
242
set current position to intersection of L1 and L2
return current position

FIG. 15C

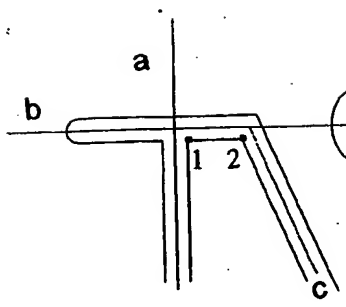


FIG. 16A

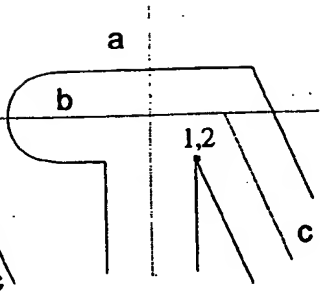


FIG. 16B

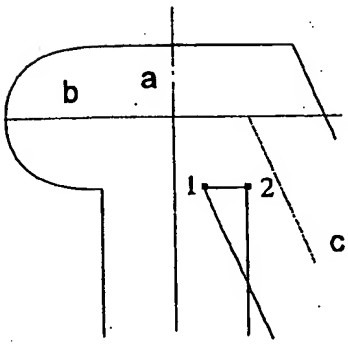


FIG. 16C

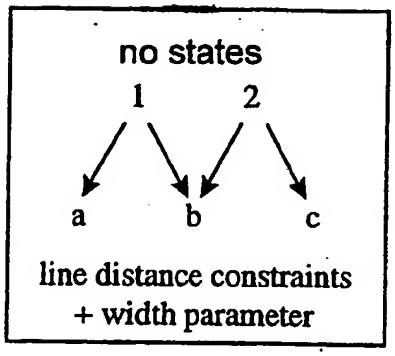


FIG.16D

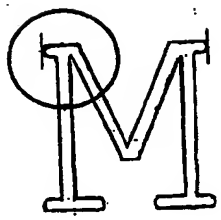


FIG. 16E

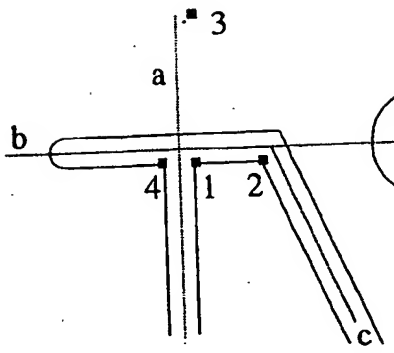


FIG. 17A

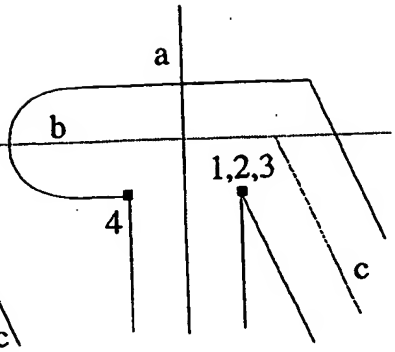


FIG. 17B

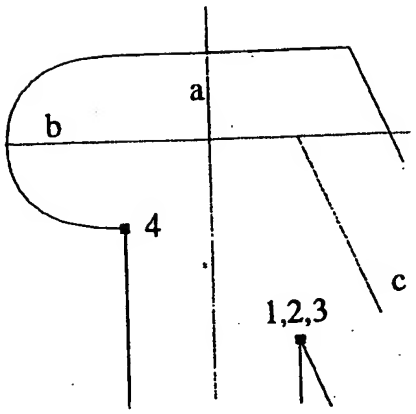


FIG. 17C

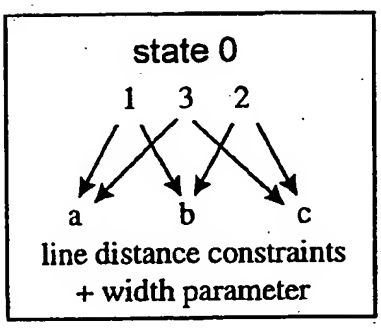


FIG. 17D

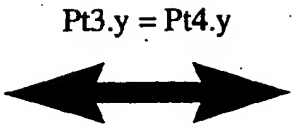


FIG. 17E

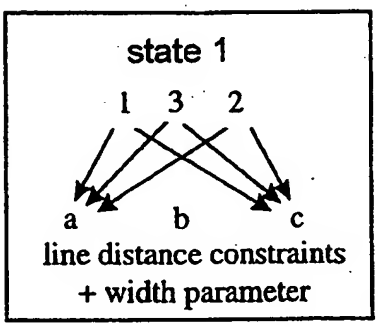


FIG. 17F

FIG. 18A

FIG. 18B

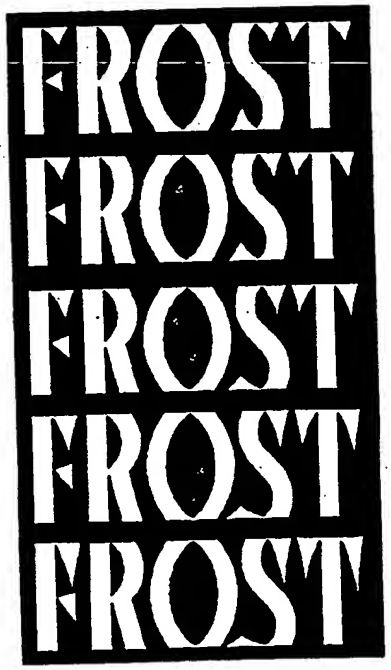


FIG. 18C

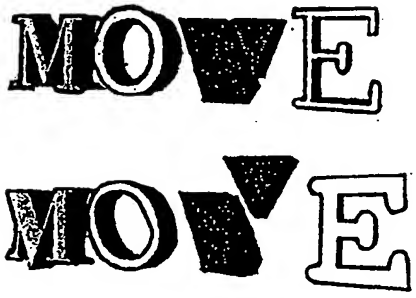


FIG. 18D

FIG. 19

FIG. 20

FIG. 21

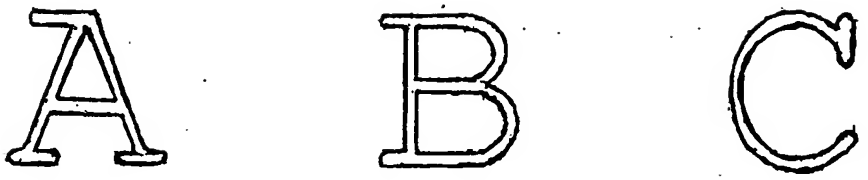


FIG. 22

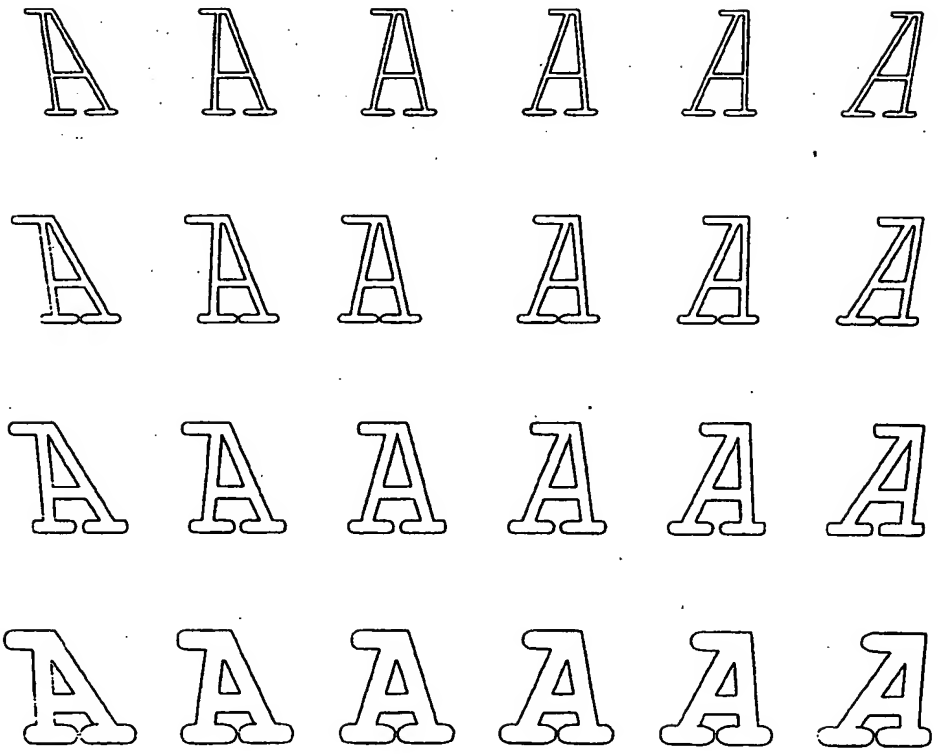


FIG. 23A

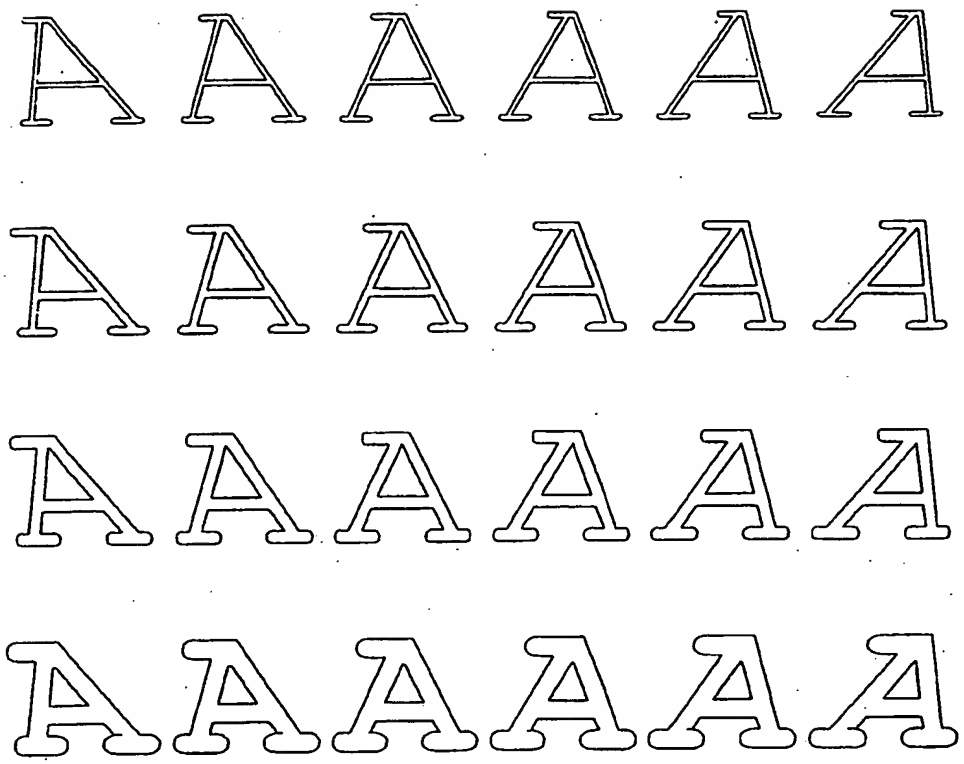


FIG. 23B

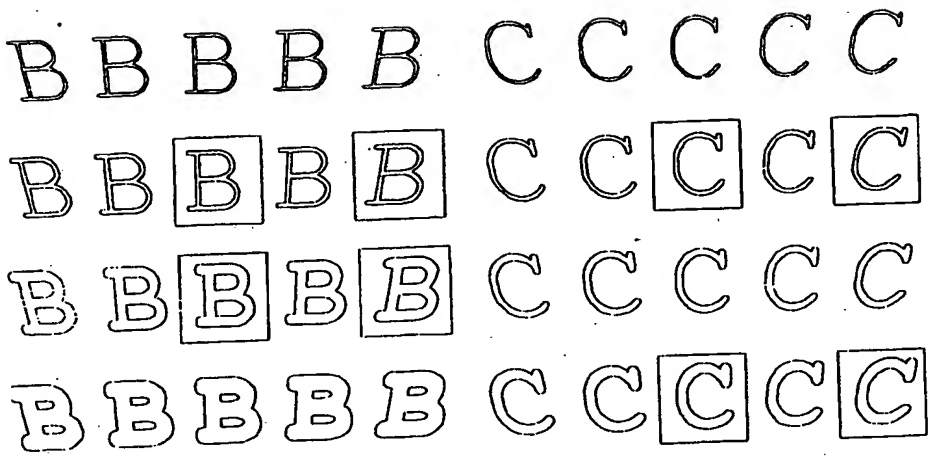


FIG. 24

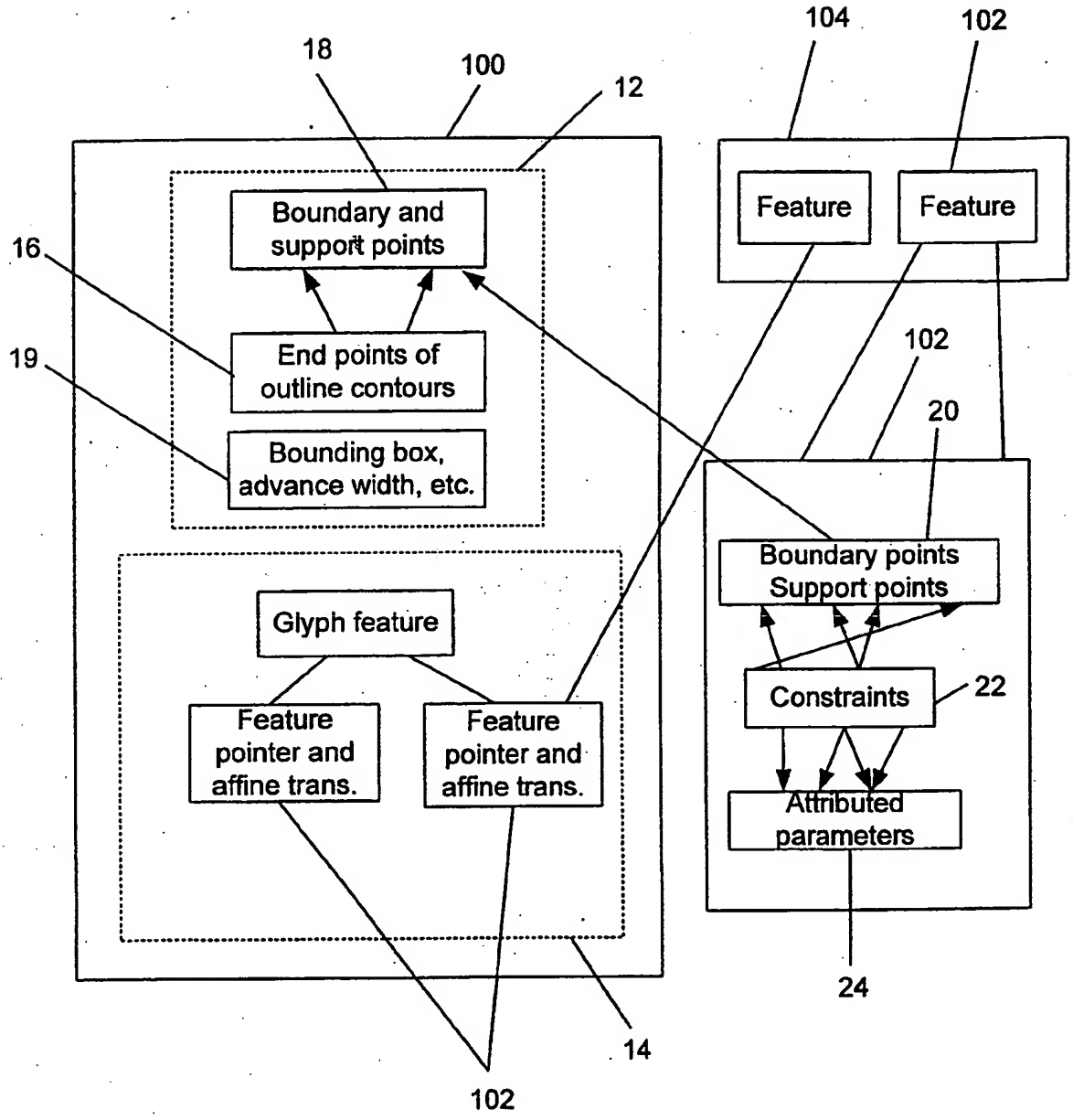

104
102
18
100
12
Feature
Feature
Boundary and support points
16
End points of outline contours
19
Bounding box, advance width, etc.
102
20
Boundary points Support points
Glyph feature
Constraints
22
Feature pointer and affine trans.
Feature pointer and affine trans.
Attributed parameters
24
14
102

FIG. 25

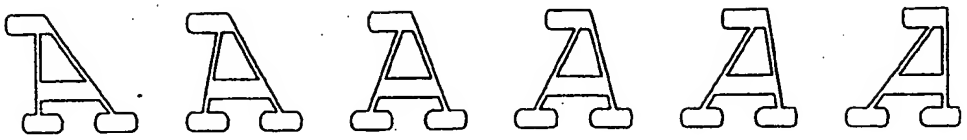


FIG. 26

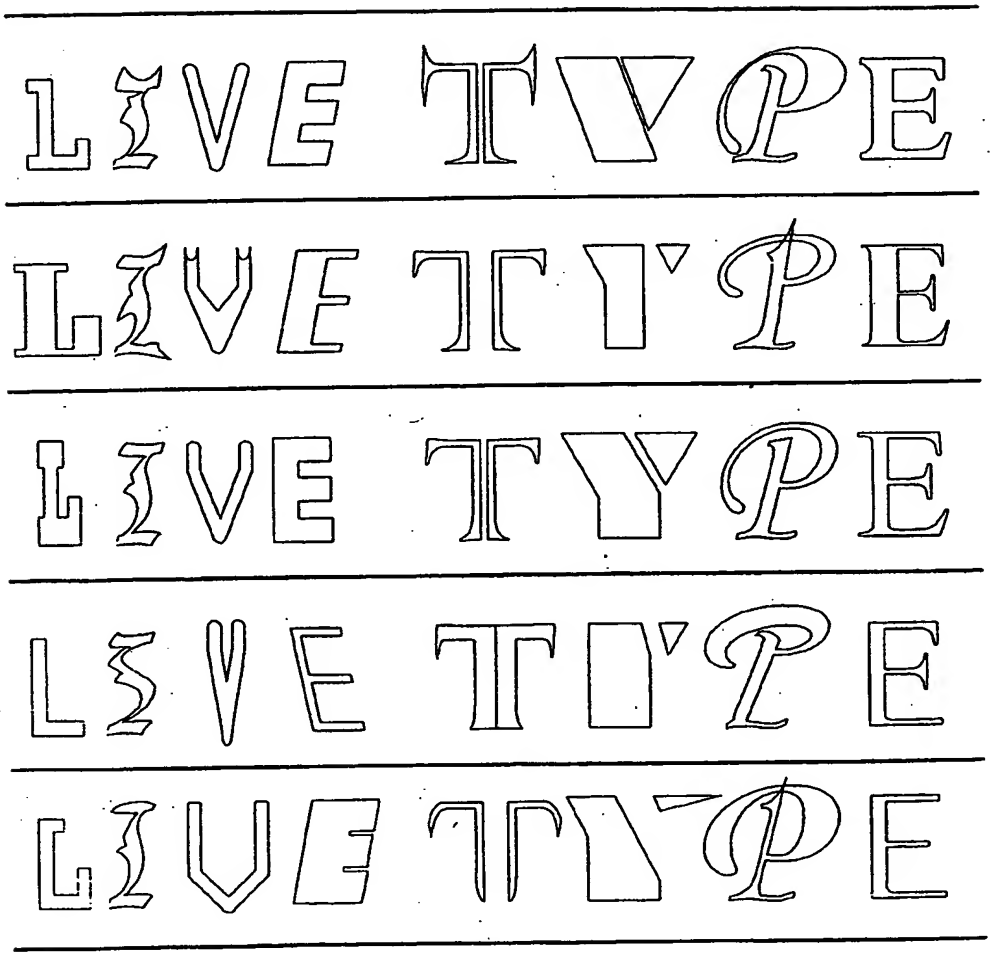


FIG. 27